Vol.2012-ARC-202 No.19
Vol.2012-HPC-137 No.19
2012/12/14

ジッタの影響を緩和する集団通信アルゴリズム

松本 英樹[1,a)]　須田 礼仁[1,2,b)]

概要：ジッタの影響により, 大規模計算機における並列計算のスケーラビリティが制限されるという問題がある. ジッタは OS のページフォールトや割り込み処理, パケットの輻輳により生じる遅延のことである. プロセッサ数が増えると, ジッタの影響は大きくなる. ジッタの影響を強く受ける allreduce について, ジッタの影響を緩和する allreduce のアルゴリズムを提案する. この手法は, バタフライと呼ばれる allreduce アルゴリズムの通信の独立性を生かした方法である. 従来のデータ交換の後に, さらにデータ交換を複数回行い, 早く届いたデータを採用することでジッタの影響を小さくする. シミュレータを作成してこの提案手法を評価した. プロセッサ数, ジッタの周期やその長さなどを変えながら評価した. ジッタの長さが 1 対 1 通信の時間よりも長く，かつプロセッサ数が 2^7 以上の場合に，従来手法はジッタの影響を特に大きく受けるが，提案手法ではそれを改善することができる．本論文中でシミュレーションした条件下では最大で従来手法のおよそ 10 倍の性能向上を得た．しかし，プロセッサ数が 2^{15} 以上になると提案手法でもジッタの影響を大きく受けることが判明した.

1. はじめに

大規模並列計算機において, 並列計算機のスケーラビリティがジッタによって制限されるという問題がある [10]. ジッタとは, オペレーティングシステムの処理やデータの通信の際に生じる遅延時間のことである. ノイズとも呼ばれ, 2 種類のジッタがある. 1 つはオペレーティングシステムの割り込み処理や入出力処理, ページフォールトによって発生するものである. もう 1 つはネットワークでパケットの輻輳によって発生するものである. 使用するプロセッサ数を, 100 万, 1000 万などと増やせば増やすほど, ジッタの影響が大きくなりパフォーマンスを大きく低下させる [10]. このような問題は以前から認識されており, 多くの調査, 手法が提案されてきた [10]. 例えば, ジッタをモデル化した論文が複数ある [3], [7], [11], [12], [13]. [12] はジッタの長さの最大値を求めてモデル化した. 他に, ジッタのシミュレータも作られている [7]. [7] は, OS のジッタだけでなく, ネットワークで生じるパケットの輻輳もシミュレートしている. さらに, ジッタの特性（周期や長さ）について詳しく研究されており [6], ジッタは周期的に発生することがわかっている [1], [9]. カーネルが発生源のジッタを発生させてジッタが影響を調べる研究もなされた [4]. ジッタによるパフォーマンスの低下を防ぐ方法も考えられてきた. 例えば, オペレーティングシステムが発生させる割り込み処理を減らすという方法がある [12]. オペレーティングシステム上で動いているプロセスの中には, 実行したいプログラムを動作させる上で, 関係のないものもある. それらを消すことで, 不要な割り込み処理の発生を減らすことで, ジッタの影響を小さくすることができる [12]. 他にもさまざまな方法があるが, その点については第 2 節でもう少し紹介する.

本論文では, アルゴリズム的側面からの解決手法を提案する. ジッタの影響は集団通信アルゴリズムに大きな影響を与えることがわかっているので [2], ジッタの影響を緩和する集団通信アルゴリズムを提案する. 以下第 2 節では, 既存研究についてもう少し詳しく述べる. 第 3 節では集団通信アルゴリズムの従来手法と提案手法について説明し, 第 4 節では実験結果を示す. 第 5 節では結論, 第 6 節では今後の課題について述べる.

2. 背景

このジッタ問題について, これまで様々な解決手法が考案されてきた. 大別すると, OS などのシステム側からの解決手法と, アルゴリズム側からの解決手法がある.

A. システム的側面による解決手法

前節で, 不要なプロセスを止めジッタの発生を減らす手法を紹介した [12]. 同様のやり方で不要なデバイス

[1] 東京大学情報理工学系研究科
Gradeuate School of Information science and Technology, the University of Tokyo
[2] JST CREST
CREST, JST
[a)] detatr@is.s.u-tokyo.ac.jp
[b)] reiji@is.s.u-tokyo.ac.jp

を外す [12] ということも考えられる．これも, 不要な割り込み処理を発生させないためである．また, デーモンを 1 つのコアで動作させ, アプリケーションはその他のコアで動作させるという方法である [10]．他には, デーモンを同期させるという手法もある [10]．各プロセスが, 同期をとってから次の同期を取るまでの間に, 処理に時間のかかるプロセッサが 1 つでもあると, 他のプロセッサはその処理が終了するのを待たなくてはならない．しかし, この手法 [10] では, デーモンの処理を度々起こすのではなく, 同じタイミングでデーモンを処理することにより, 全体の実行時間を短縮している．

B. アルゴリズム的側面による解決手法
アルゴリズム的側面からのアプローチに, ノード内で動的に負荷を分散させるという方法がある [8].

ジッタ問題に対して, OS などのシステム的側面からのアプローチが多数見受けられたが, 一方でアルゴリズムによるアプローチはあまりなかった．そこで, 本研究ではアルゴリズム的側面からの手法について研究する．集団通信特に allreduce により, パフォーマンスが大きく低下するので [10], allreduce について, ジッタの影響を緩和するアルゴリズムを提案する．

3. 提案手法

使用するプロセッサ数は 2 の冪乗とし, 通信時間は $\alpha+\beta N$ とする．ただし, α はネットワークレイテンシー, β はネットワークバンド幅, N は, データサイズを表す．また, ネットワークのトポロジーは考慮しない．さらに, 使用する計算機はフルバイセクションであることを仮定する．

3.1 従来の計算方法

Allreduce のアルゴリズムには様々な方法があるが, 本研究ではバタフライと呼ばれるアルゴリズムに対して, ジッタの影響を緩和するアルゴリズムを提案する．バタフライというアルゴリズムは, 各プロセッサが自分の持っているデータの交換および演算を繰り返すアルゴリズムであり, 最終的に各プロセッサは同じ計算結果を得る．例を**図 1** に示す．これは allreduce の計算の流れを表したものである．ここでは各プロセッサのもつ値の和を得る計算を行っている．横に並んだ 4 つの長方形は 4 つのプロセッサを, 各長方形の中の数字は各プロセッサが持っているデータを, 矢印はデータの通信を表している．以下, 左側のプロセッサから順に P1, P2, P3, P4 と呼ぶことにする．各プロセッサのデータの更新と通信の様子は上から下に向かって変化している．なお, 各プロセッサは他のプロセッサからデータを受け取った際, すでに持っていたデータとの和をとるが, わかりやすさのため計算結果ではなく計算式を書いている．プロセッサが 2^K 個ある場合, 各プロセッサは $\log 2^K = K$ 回だけデータを交換すればよい．この場合, プロセッサ数が 4 個なので各プロセッサは $\log 2^2 = 2$ 回だけデータ交換を行えばよい．

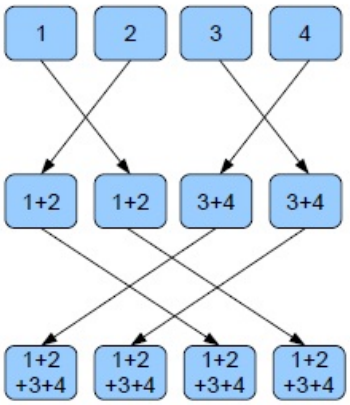

図 1 allreduce のバタフライアルゴリズム
Fig. 1 the butterfly algorithm of allreduce

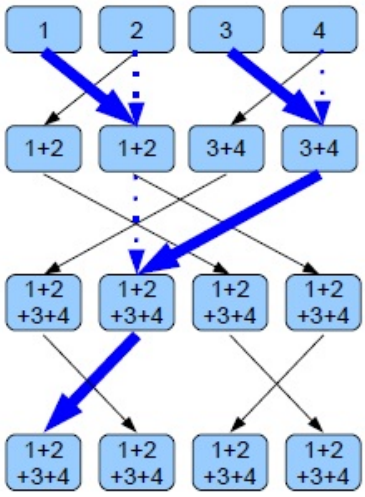

図 2 提案手法を用いたときの,3 回目のデータ交換で P1 が P2 から受け取るデータの流れ
Fig. 2 In case of the proposed method, the data flow which P1 receive data from P2 in third data exchange

3.2 提案手法

本論文で提案する手法は, 先に紹介したバタフライと呼ばれる従来のアルゴリズムに対して, データの交換をさらに行うことでジッタの影響を緩和する．プロセッサ数が 2^K 個ある場合, K 回のデータの交換を終えた後, さらにデータの交換を行うのである．たとえば, **図 2** のような場合を考える．これは, 従来手法を表した図 1 と同様に, 4 つのプロセッサが allreduce の計算を行うときの計算の流れを表したものであるが, 図 1 よりもデータの交換が 1 回分多い．図 2 における 1 回目と 2 回目の交換は図 1 と同じであるが, 3 回目の交換を新たに加えている．ここで, プロセッサ数が 4 つの場合, 提案手法がジッタの影響をどのように緩

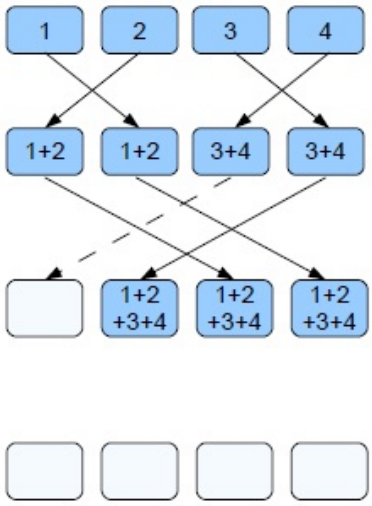

図 3 2 回目のデータ交換において, ジッタによって P1 は P3 が送信したデータをまだ受信できていない

Fig. 3 In the second data exchange, P1 has not received the data P3 send because of the jitter

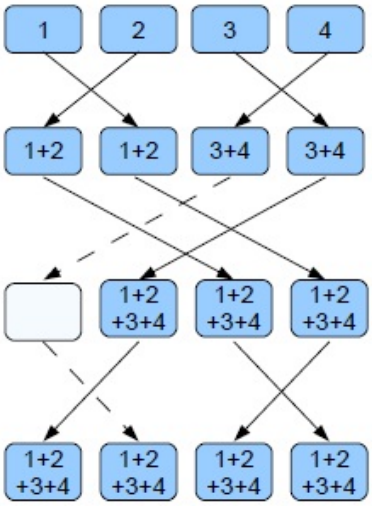

図 4 P3 が送信したデータよりも P2 が送信したデータの方が速く P1 に届いた.

Fig. 4 In case of proposed method, the data the P2 send arrived at P1 faster than that of P3

和するのかを示す. 1 回目のデータ交換ではジッタの影響を受けず, 2 回目のデータの交換においてジッタによる遅延が生じたとする（**図 3**）. P1 は P3 から送られるデータを待たなければならない. 一方, P1 はすでに 2 回目のデータの受信および計算を終えていたならば, その結果を, まだ計算を終えられていないプロセッサに送信することができる. ここで 3 回目のデータの交換をする. P1 は自身の計算を終えられていないので, P2 にデータを送信することはできないが, P2 は P1 にデータを送信することができる. もし, P3 からのデータよりも, P2 からのデータが早く届いた場合, 従来手法よりも早く計算できる（**図 4**）. この例では, 従来手法に対してデータの交換を 1 回増やしたが, プロセッサ数が 4 つの場合はさらにもう 1 回, すなわち合計 2 回のデータの交換を行うことができる.

ここで図 2 の最後のデータ交換は, 最初のデータ交換と同じプロセッサの組, つまり偶数番目と奇数番目のプロセッサが対になって行うことが重要である. 図 1 のアルゴリズムでは, 最初のデータ交換以外では偶数番目のプロセッサと奇数番目のプロセッサはデータ交換をしない. すなわち, 最初のデータ交換を除けば, 偶数番目のプロセッサはそれらだけで通信および計算を完結しており, 奇数番目のプロセッサがジッタの影響を受けたとしても, 偶数番目のプロセッサがその影響を受けることはない. 言い換えれば, バタフライアルゴリズムの 2 回目のデータ交換以降は, 2 つの独立な allreduce になっているのである. したがって, ジッタの影響を受けていない偶数番目のプロセッサが, ジッタの影響を受けた奇数番目のプロセッサにデータを送信することによって, （ジッタの影響がない場合の実行時間）+（1 回の通信時間）で allreduce を完了することができる. ここで注意したいのは, このような状況は, 各プロセッサの最初の冗長なデータ交換の通信相手は, バタフライアルゴリズムの最初のデータ交換の通信相手と同一の場合にのみ達成されるため, そのように通信するプロセッサを選択しなければならないということである.

また同様に, 冗長なデータ交換を 2 回行う場合は, バタフライアルゴリズムの最初の 2 回分のデータ交換に相当する通信パターンで, 冗長なデータ交換を行う必要がある. バタフライアルゴリズムの 3 回目のデータ交換は, 4 つの独立な allreduce からなっているからである. そのため, プロセッサ番号が 4 で割って m（m = 0, 1, 2, 3）余るプロセッサ集合のうち, ひとつの集合だけれもジッタの影響を受けなければ, そのプロセッサ集合から他のプロセッサ集合にデータを送信することができる.

ジッタが影響を及ぼさない場合は, もちろん従来手法の方が速い. プロセッサ数が 2^K 個ある場合, K 回のデータの交換を終えれば所望のデータが得られる. わざわざ $K+1$ 回目のデータ交換を行う必要はない. しかし図 3 で示した場合のように, ジッタの影響により遅延したデータを待つよりも, 他のプロセッサが速く計算を終えていれば, その結果を送信してもらえばよい. 受信するのが遅いデータを待つよりも, 早く受信したデータをもらうというのが提案手法の方針である.

4. シミュレーションによる予備評価

本研究では計算時間をシミュレーションして実験を行った. ジッタの頻度や長さの違いによる, 提案手法と従来手法の実行時間を比較した.

表 1 シミュレーションの予備評価で用いる記号とその意味
Table 1 symbols and the meanings used for the exploratory evaluation of the simulation

記号	意味	単位
α	ネットワークレイテンシー	秒
β	ネットワークバンド幅	秒
γ	メモリバンド幅	秒
P	プロセッサ数	-
N	データサイズ	バイト
period	ジッタの発生周期	秒
duration	ジッタの長さ	秒
Tcomm（$=\alpha+\beta N$）	通信時間	秒
T	追加したデータ送信回数	-

4.1 シミュレーションモデル

4.1.1 パラメータ

本シミュレーションで用いる各パラメータを**表 1** に示す. ただし, ネットワークレイテンシーは 1μ（sec）, ネットワークバンド幅は 1（GB）, メモリバンド幅は 10（GB）である計算機環境を仮定している. なお, 計算機のメモリバンド幅およびネットワークバンド幅, ネットワークレイテンシーを固定している.

4.1.2 実行時間の計算方法

基本的には, 4.1.1 で定めた値を用いてプロセッサごとに, 最終的な答えを得るまでの時間を求め, それらのうち最大となる値を allreduce の実行時間とする. プロセッサごとの計算時間を求めるときは, プロセッサ数を 2^K 個用いている場合, データの交換を $\log 2^K = K$ 回行う間と, その後に追加したデータの交換をする間は計算の仕方が異なる.

A. データの交換が 1 回以上 K 回以下の場合
データの受信時間および演算時間を, 毎回計算し, 累計する.

- データの受信に要する時間

$$\alpha + \beta \times N + \texttt{Network-noise} \quad (1)$$

- データの演算に要する時間

$$\gamma \times N + \texttt{OS-jitter} \quad (2)$$

なお, Network-noise および OS-jitter は発生したジッタによる遅延時間を表しており, それぞれネットワークで発生するジッタによる遅延時間, オペレーティングシステムが発生させるジッタによる遅延時間を表す. ジッタについては後述する.

B. データの交換が K+1 回以上の場合
指定した回数分（T）, 最終的なデータを指定されたプロセッサに送信する. 各プロセッサに届くデータは, 従来手法による K 回目の交換により届くデータと,K+1 回以上のデータ交換により届く複数のデータ（T 個のデータ）があるが, これらのうち最も早く届いたデータを採用する. データの演算時間は 0 である. K 回のデータ交換を終えたプロセッサは最終的なデータを得ているので, あとは他のプロセッサにデータを送信するだけである. そのため, OS が発生させるジッタの影響はないが, ネットワークで発生するジッタの影響はあり. その値は式（1）によって計算する.

提案手法では T の値により, 最短時間は異なる. プロセッサ数が $P = 2^K$ 個のとき, T の最大値は $\log 2^K = K$ である. 本シミュレーションでは各 T の値について 30 回のシミュレーションの平均値を求める. それらの平均値, すなわち T を 1 から K まで変えたときの最小値を実行時間とする. なお, 本シミュレーションでは, ジッタの影響度合いを調べるため, ジッタが発生しないない場合についても実行時間を計算している. ジッタは発生しないときは式（1）の `Network-noise` および式（2）の `OS-jitter` を 0 にする.

4.1.3 ジッタのモデル

先に述べたように, ジッタには 2 種類あり, オペレーティングシステムにより生じるものと, ネットワークで生じるものがある. 本研究で用いたシミュレータでは, OS ジッタを周期的に発生させるものとし, ネットワークで生じるジッタはポアソン分布に従うものとしている.

A. オペレーティングシステムが発生させるジッタ
ジッタの発生周期は各プロセッサで共通である. しかし, OS が発生させるジッタのタイミングは各プロセッサごとに異なるので, 最初に発生させるタイミングを乱数を用いて決めている.

B. ネットワークで生じるジッタ
ジッタの発生周期を period とすると, ネットワークで生じるジッタは平均待ち時間 1/period であるポアソン分布に従い発生する.

4.2 実験結果

A. まず, 通信時間とジッタの周期の比 (Tcomm/period) とプロセッサ数の値を変化させたときの, ジッタの効果, 提案手法の効果およびそのときの冗長なデータ送信の回数の最適値をグラフに示す. ジッタの周期を 1.0e-3(秒), ジッタの長さを 1.0e-5(秒) に固定したときの, ジッタの効果を**図 5**, 提案手法の効果を**図 6** に, そしてそのときの冗長なデータ送信の回数の最適値を**図 7** に示す. なお, 各グラフ横軸は通信時間をジッタの長さで割った値の対数表示, 縦軸はプロセッサ数の対数表示となっている.

図 5 より, ジッタの長さよりも通信時間の方が長い場合はジッタの影響が小さい. しかし, ジッタの長さよりも通信時間の方が短い場合はジッタの影響は大きくなる. これは, ジッタの長さが通信時間よりも小さいときはジッタが通信時間に隠れることができるが, 逆の場合はジッタが通信時間に隠れず, 計算時間に大きな遅延を生じることになるからだと考えられる. こ

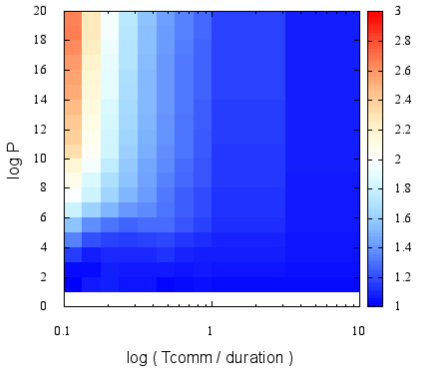

図 5 ジッタの影響
（ジッタありの従来手法の計算時間）/（ジッタなしの従来手法の計算時間）
（period=1.0e-3, duration=1.0e-5）

Fig. 5 the influence of the jitter
（the computation time of the butterfly algorithm with jitter）/（the computation time of the butterfly algorithm with jitter）
（period=1.0e-3, duration=1.0e-5）

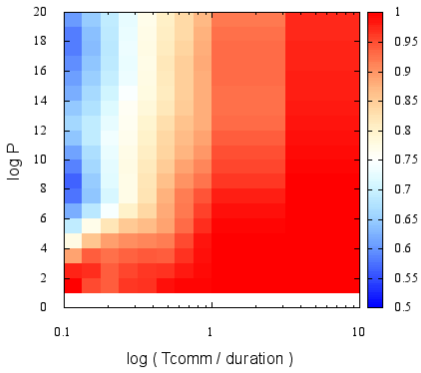

図 6 従来手法と比較した提案手法の効果
（提案手法の計算時間）/（従来手法の計算時間）
（period=1.0e-3, duration=1.0e-5）

Fig. 6 the effectiveness of the proposed algorithm compared to the butterfly algorithm.
（the computation time of the proposed algorithm）/（the computation time of the proposed algorithm）
（period=1.0e-3, duration=1.0e-5）

のことは [10] の指摘と一致している．

図 6 を見ると，ジッタの長さよりも通信時間の方が長い場合，提案手法の効果は小さい．一方，ジッタの長さよりも通信時間の方が短い場合，提案手法の効果は大きくなる．この傾向は先の図 5 のグラフと似ている．これらより，ジッタの影響が大きいときに提案手法の効果が高いと言える．

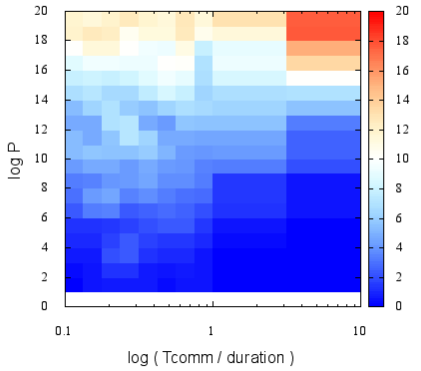

図 7 実行時間が最短であるときの，冗長なデータ送信の回数の最適値
（period=1.0e-3, duration=1.0e-5）

Fig. 7 the numbers of times of redundant data sending when the computation time is shortest）
（period=1.0e-3, duration=1.0e-5）

最後に図 7 で計算時間を最小にする T（追加したデータの送信回数）を確認すると，プロセッサ数が増えると最適な T も大きくなるということがわかる．

B. 通信時間とジッタの周期の比 (Tcomm/period) とプロセッサ数の値を変化させたときの，ジッタの効果，提案手法の効果およびそのときの冗長なデータ送信の回数の最適値をグラフに示す．ここではネットワークレイテンシー α を 1.0e-7(秒) とし，ジッタの発生周期を 1.0e-2 秒，ジッタの長さを 1.0e-4 秒とする．このときの，ジッタの影響の強さを**図 8**，提案手法の効果を**図 10**，そのときの冗長なデータ送信の回数を**図 11** に示す．

図 8 では，左上端を除くと全体的にジッタの影響が一様に見える．この図 8 の z 軸の範囲は 1 から 20 までであるが，それを 1 から 5 までに設定したものを**図 9** に示す．z の値が 5 倍以上は赤色になっている．図 5 の場合と同様に，ジッタの長さよりも通信時間の方が長い場合はジッタの影響が小さく，ジッタの長さよりも通信時間の方が短い場合はジッタの影響は大きいことがわかる．原因も同じく，ジッタの長さが通信時間よりも小さいときはジッタが通信時間に隠れることができるが，逆の場合はジッタが通信時間に隠れず，計算時間に大きな遅延を生じることになるからだと考えられる．図 5，図 8 を比較してみると，図 8 の方がジッタの影響が大きいと言える．ネットワークレイテンシーを 10 分の 1 にするとジッタの影響はおよそ 10 倍にまで大きくなることがわかる．さらに，(Tcomm/duration) の値が 0.01 よりも小さいとき，その影響は顕著になる．

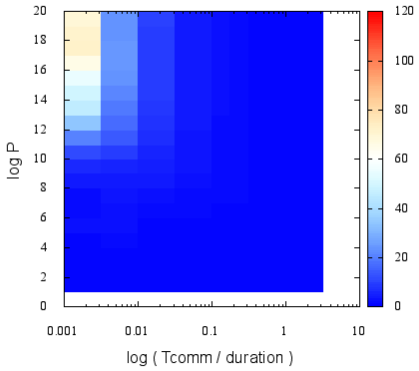

図 8 ジッタの影響（Z 軸の範囲 [0,120]）
（ジッタありの従来手法の計算時間）/（ジッタなしの従来手法の計算時間）
（period=1.0e-2, duration=1.0e-4）

Fig. 8 the influence of the jitter（the z range [0,120]）
（the computation time of the butterfly algorithm with jitter）/（the computation time of the butterfly algorithm with jitter）
（period=1.0e-2, duration=1.0e-4）

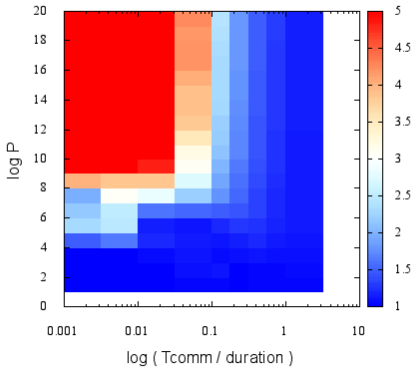

図 9 ジッタの影響（Z 軸の範囲 [1,5]）
（ジッタありの従来手法の計算時間）/（ジッタなしの従来手法の計算時間）
（period=1.0e-2, duration=1.0e-4）

Fig. 9 the influence of the jitter（z range [1,5]）
（the computation time of the butterfly algorithm with jitter）/（the computation time of the butterfly algorithm with jitter）
（period=1.0e-2, duration=1.0e-4）

このことは, 非常に大規模な計算機では, ネットワークレイテンシーを 10 分の 1 に短縮しても, ジッタの影響により所要時間はそれほど短縮できない危険性があるということを示している.

図 10 を見ると, 通信時間がジッタの長さと同じぐらいである場合, プロセッサ数に関わらず提案手法の効果があまり良くない. 図 6 では, ジッタの影響が大きい場合には, 提案手法の効果も大きかった. プロセッサ数が 2^8 より大きく, かつ通信時間がジッタの長さの 10 分の 1 より小さい場合に, 提案手法の効果が大きい. 特にプロセッサ数が 2^{10} から 2^{14} であり, かつ通信時間がジッタの長さの 100 分の 1 より小さい場合に効果が特に大きいことがわかる. これを調べるために, (Tcomm/duration)=1.0e-2 であるときの, プロセッサ数に対する計算時間の変化を**図 12** に示す. プロセッ

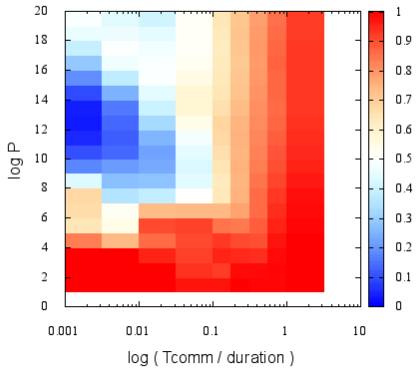

図 10 従来手法と比較した提案手法の効果
（提案手法の計算時間）/（従来手法の計算時間）
（period=1.0e-2, duration=1.0e-4）

Fig. 10 the effectiveness of the proposed algorithm compared to the butterfly algorithm.
（the computation time of the proposed algorithm）/（the computation time of the proposed algorithm）
（period=1.0e-2, duration=1.0e-4）

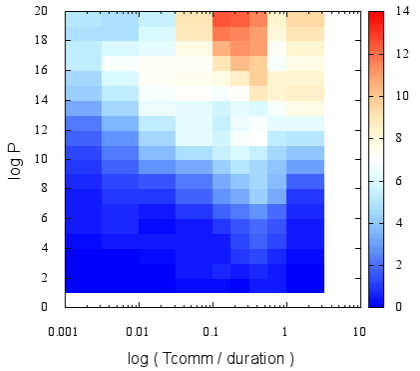

図 11 実行時間が最短であるときの, 冗長なデータ送信の回数の最適値
（period=1.0e-2, duration=1.0e-4）

Fig. 11 the numbers of times of redundant data sending when the computation time is shortest）
（period=1.0e-2, duration=1.0e-4）

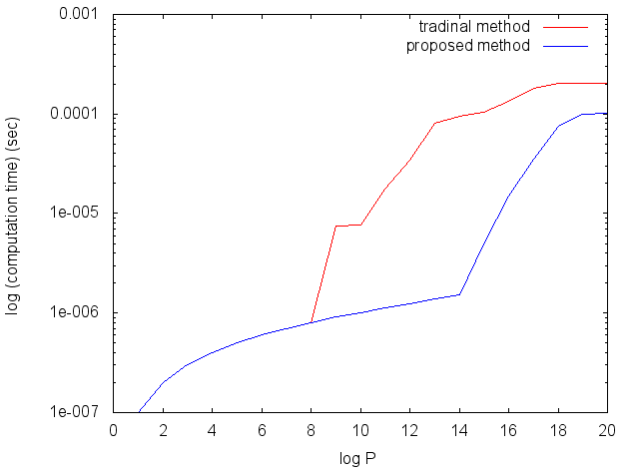

図 12 従来手法の計算時間と提案手法の計算時間の最小値
（Tcomm/duration=1.0e-2, period=1.0e-3, duration=1.0e-5）

Fig. 12 the computationtime of the traditional method and the shortest computation time of the proposed method
（Tcomm/duration=1.0e-2, period=1.0e-3, duration=1.0e-5）

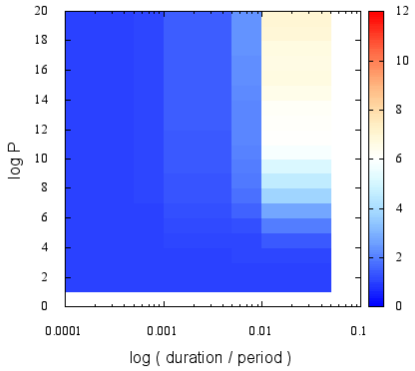

図 13 ジッタの影響（Z 軸の範囲 [1,12]）
（ジッタありの従来手法の計算時間）/（ジッタなしの従来手法の計算時間）
（period=1.0e-3, N=8）

Fig. 13 the influence of the jitter（z range [1,12]）
（the computation time of the butterfly algorithm with jitter）/（the computation time of the butterfly algorithm with jitter）
（period=1.0e-3, N=8）

サ数が 2^7 から 2^{14} の場合は, 従来手法に比べ提案手法の方が実行時間が大変小さく, プロセッサ数が 2^{14} よりも大きくなると提案手法の計算時間が急激に大きくなるため, 図 10 ではプロセッサ数が 2^{10} から 2^{14}, 通信時間がジッタの長さの 100 分の 1 より小さい場合に大きな効果が得られる結果となっている. すなわち, 2^{15} 以上のプロセッサ数に対しては, 提案手法を用いてもジッタの影響を完全には排除できないことがわかった. 図 10 と図 6 を比較すると, ネットワークレイテンシーを小さくした図 10 の方がジッタの影響が大きい分, 提案手法の効果も大きいということが読み取れる.

冗長なデータ交換回数の最適値を図 11 で確認すると, プロセッサ数が同じとき, 通信時間がジッタの長さよりも小さいほど, より少ない回数の冗長なデータ交換で効果が出る傾向がある.

C. ここではネットワークレイテンシーを元に戻し, α=1.0e-6(秒) とする. ジッタの長さとジッタの周期の比 (duration/period) とプロセッサ数の値を変化させたときの, ジッタの影響力, 提案手法の効果およびおよび実行時間が最短となるときの冗長なデータ送信回数を示す. ジッタの周期を 1.0e-3 (秒) に固定したときの, ジッタの影響力を**図 13**, 提案手法の効果**図 15**, および実行時間が最短となるときの冗長なデータ送信回数の最適値を**図 16** に示す. また, ジッタの周期を 1.0e-2 (秒) に固定したときの, ジッタの影響力を**図 17**, 提案手法の効果を**図 19**, および実行時間が最短となるとき

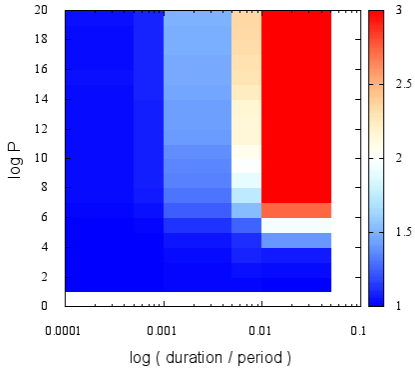

図 14 ジッタの影響（Z 軸の範囲 [1,3]）
（ジッタありの従来手法の計算時間）/（ジッタなしの従来手法の計算時間）
（period=1.0e-3, N=8）

Fig. 14 the influence of the jitter（z range [1,3]）
（the computation time of the butterfly algorithm with jitter）/（the computation time of the butterfly algorithm with jitter）
（period=1.0e-3, N=8）

の冗長なデータ送信回数の最適値を**図 20** に示す. なお, データサイズは 8 バイトに固定している.

図 13, 図 17 は, ジッタの効果を表すグラフであるが, いずれも右上端を除くと全体的にジッタの影響が一様に見える. そこで z 軸の上限を小さい値に設定したグ

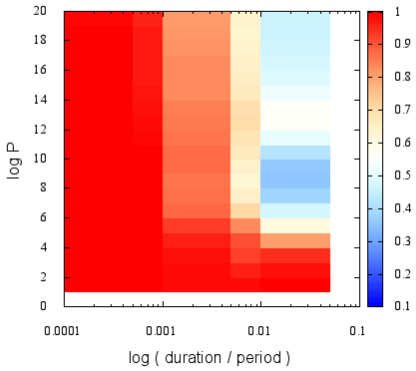

図 15 従来手法と比較した提案手法の効果
（提案手法の計算時間）/（従来手法の計算時間）
（period=1.0e-3, N=8）

Fig. 15 the effectiveness of the proposed algorithm compared to the butterfly algorithm.
（the computation time of the proposed algorithm）/（the computation time of the proposed algorithm）
（period=1.0e-3, N=8）

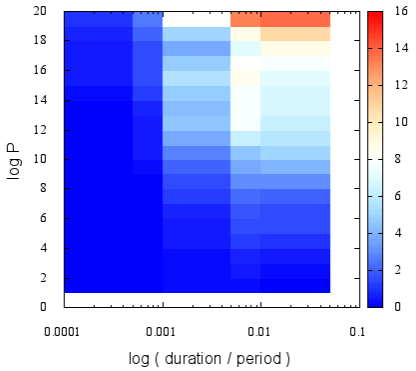

図 16 実行時間が最短であるときの, 冗長なデータ送信回数の最適値
（period=1.0e-3, N=8）

Fig. 16 the numbers of times of redundant data sending when the computation time is shortest）
（period=1.0e-3, N=8）

ラフを示す. 図 13 について, z 軸の範囲を 1 から 3 までに設定したものを**図 14** に示す. また, 図 17 について, z 軸の範囲を 1 から 5 までに設定したものを**図 18** に示す. z の値が上限を超えた場合は赤色になっている. 図 13, 図 17, 図 14, 図 18 より, ジッタの長さがジッタの周期の 100 分の 1 よりも小さい場合はジッタの影響は小さいが, 100 分の 1 よりも大きくい場合はジッタの影響が大きくなる.

また図 15, 図 19 より, ジッタの影響の大きい場合

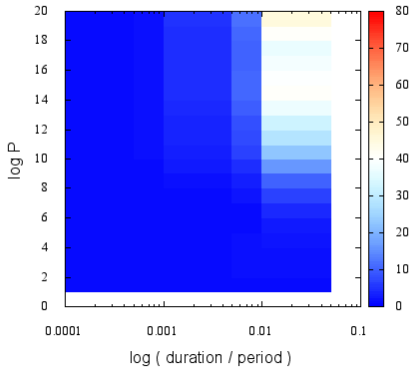

図 17 ジッタの影響（Z 軸の範囲 [0,80]）
（ジッタありの従来手法の計算時間）/（ジッタなしの従来手法の計算時間）
（period=1.0e-2, N=8）

Fig. 17 the influence of the jitter（z range [0,80]）
（the computation time of the butterfly algorithm with jitter）/（the computation time of the butterfly algorithm with jitter）
（period=1.0e-2, N=8）

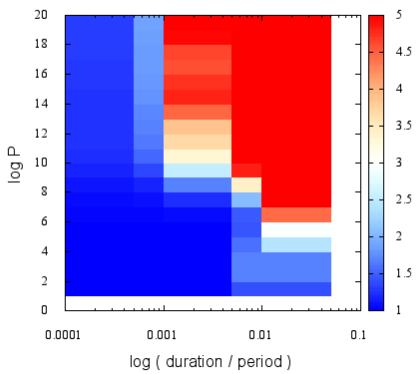

図 18 ジッタの影響（Z 軸の範囲 [1,5]）
（ジッタありの従来手法の計算時間）/（ジッタなしの従来手法の計算時間）
（period=1.0e-2, N=8）

Fig. 18 the influence of the jitter（z range [1,5]）
（the computation time of the butterfly algorithm with jitter）/（the computation time of the butterfly algorithm with jitter）
（period=1.0e-2, N=8）

には, 提案手法の効果が大きい. ただし, ジッタの長さがジッタの周期の 100 分の 1 よりも大きく, かつプロセッサ数が 2^7 から 2^{14} の場合, 提案手法の効果は大きくなる. **図 21** は (duration/period)=0.01 であるときの, プロセッサ数に対する計算時間の変化を表したグ

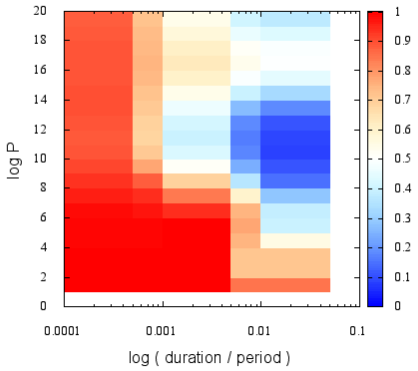

図 19 従来手法と比較した提案手法の効果
（提案手法の計算時間）/（従来手法の計算時間）
（period=1.0e-2, N=8）

Fig. 19 the effectiveness of the proposed algorithm compared to the butterfly algorithm.
（the computation time of the proposed algorithm）/（the computation time of the proposed algorithm）
（period=1.0e-2, N=8）

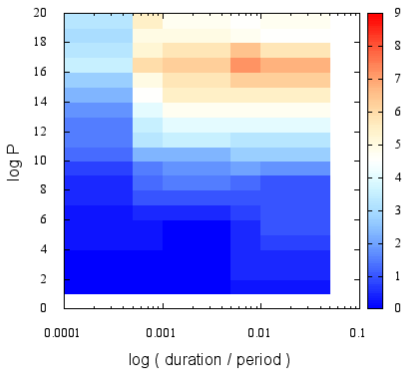

図 20 実行時間が最短であるときの, 冗長なデータ送信回数の最適値
（period=1.0e-2, N=8）

Fig. 20 the numbers of times of redundant data sending when the computation time is shortest）
（period=1.0e-2, N=8）

ラフである. プロセッサ数が 2^7 から 2^{14} の場合は, 従来手法に比べ提案手法の方が実行時間が大変小さく, プロセッサ数が 2^{14} よりも大きくなると提案手法の計算時間が急激に大きくなる. そのため, 図 19 ではプロセッサ数が 2^7 から 2^{14}, 通信時間がジッタの長さの 100 分の 1 より小さい場合に大きな効果が得られる結果となっている.

図 16, 図 20 から, 計算時間が最小となるときの T の値を確認すると, プロセッサ数が多くなると, 実行時間を少しでも短くしようとすると, 冗長なデータ送信回数（T）を大きい値に設定しなければならない.

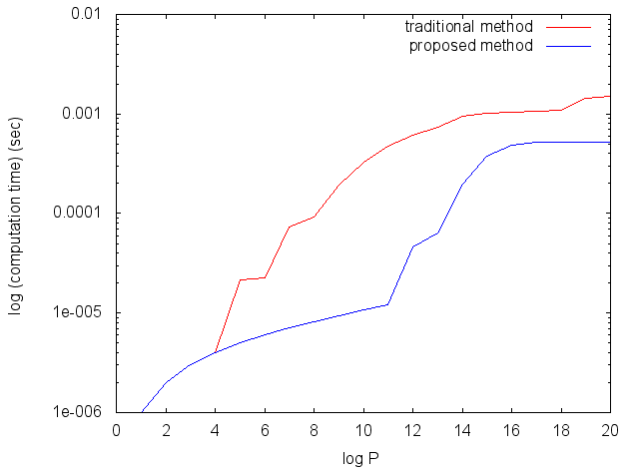

図 21 従来手法の計算時間と提案手法の計算時間の最小値
（duration/period=1.0e-2, period=1.0e-3, duration=1.0e-5, N=8）

Fig. 21 the computationtime of the traditional method and the shortest computation time of the proposed method
（duration/period=1.0e-2, period=1.0e-3, duration=1.0e-5, N=8）

5. まとめ

OS やネットワークを発生源とするジッタにより, 大規模計算機での並列計算のスケーラビリティを制限される. この問題に対して, システムレベルでの対処法などは多く研究されてきたが, アルゴリズムによる手法はあまりなされていきていない. そこで本研究では, このジッタの影響を緩和する allreduce アルゴリズムを提案した. 本手法は, 通常のバタフライアルゴリズムを実行した後に, 冗長なデータ交換を行い, 計算するよりも早くデータが届いた場合にそれを採用するというものである. シミュレータを作成し, 従来の allreduce と提案手法による allreduce の実行時間を比較した. その結果, 通信時間よりもジッタの長さの方が長くジッタの影響が大きい場合は, 提案手法の効果も大きくなり, 逆に通信時間よりもジッタの長さの方が短くジッタの影響も小さい場合には, 提案手法の効果も小さいことがわかった. また, ネットワークレイテンシーを 10 分の 1 にするとジッタの影響も 10 倍近くになるものの, その分提案手法の効果も大きくなる. またジッタの周期に対してジッタの長さが小さくジッタの影響は小さい場合, 提案手法の効果は小さく, ジッタの周期に対してジッタの長さが大きいほどジッタの影響が大きい場合は, 提案手法の効果も大きくなる.

6. 今後の課題

提案手法では, 従来の allreduce の後に冗長なデータ交換を行うが, その冗長なデータ交換の際にはデータ送信のための処理時間を 0 としている. 実際の計算機ではそのデータ送信のための処理時間が必要であるから, この時間を考慮した上で提案手法の評価を行いたい. さらに, ジッタの影響を緩和するために, 追加するデータ交換を何回ぐらい行えばよいかという基準を調べたい. 本研究では 2 の累乗個のプロセッサを用いた場合のアルゴリズムを扱ったが, それ以外のプロセッサ数を利用する場合もあるであろうから, その場合についても考える必要がある.

謝辞 本研究の一部は JST CREST「進化的アプローチによる超並列複合システム向け開発環境の創出」, 科学研究費「超高速・超低消費電力物質科学シミュレーション方式の研究開発」の援助を受けています.

参考文献

[1] Akkan, H et al.: Stepping towards noiseless Linux environment, *Proceedings of the 2nd International Workshop on Runtime and Operating Systems for Supercomputers*, pp. 7:1–7:7, (2012).
[2] Beckman, P. et al.: The Influence of Operating Systems on the Performance of Collective Operations at Extreme Scale, *2006 IEEE International Conference on Cluster Computing*, Vol. 0, pp. 1–12 (2006).
[3] De and Pradipta.: Identifying sources of Operating System Jitter through fine-grained kernel instrumentation, *Proceedings of the 2007 IEEE International Conference on Cluster Computing*, pp. 331–340, (2007).
[4] Ferreira and Kurt B. et al.: Characterizing application sensitivity to OS interference using kernel-level noise injection, *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pp. 19:1–19:12, (2008).
[5] Agarwal, S. et al.: The impact of noise on the scaling of collectives: a theoretical approach, *Proceedings of the 12th international conference on High Performance Computing*, pp. 280–289, (2005).
[6] Hoefler, T. et al.: The impact of network noise at large-scale communication performance, *Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing*, pp. 1–8, (2009).
[7] Hoefler, T. et al.: Characterizing the Influence of System Noise on Large-Scale Applications by Simulation, *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–11, (2010).
[8] Kale and Vivek.: Load balancing for regular meshes on SMPs with MPI, *Proceedings of the 17th European MPI users' group meeting conference on Recent advances in the message passing interface*, pp. 229–238, (2010).
[9] Morari and Alessandro.: A Quantitative Analysis of OS Noise, *Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium*, pp. 852–863, (2011).
[10] Petrini, F. et al.: The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q, *Extreme scale computing: Modeling the impact of system noise in multicore clustered systems*, pp. 55–, (2003).
[11] Seelam, S. et al.: Extreme scale computing: Modeling the impact of system noise in multicore clustered systems, *IEEE International Parallel & Distributed Processing (IPDPS)*, pp. 1–12, (2010).
[12] Tsafrir and Dan et al.: System noise, OS clock ticks, and fine-grained parallel applications, *Proceedings of the 19th annual international conference on Supercomputing*, pp. 303–312, (2005).
[13] Gengbin Z. et al.: Simulating Large Scale Parallel Applications using Statistical Models for Sequential Execution Blocks, *Proceedings of the 16th International Conference on Parallel and Distributed Systems (ICPADS 2010)*, pp. 10–15, (2010).