卒業研究報告書

題 目

BSP モデル上での整列法

指導教員

石水隆助手

報告者

01-1-26-007

北村 勇樹

近畿大学理工学部電気工学科

平成 17 年 2 月 19 日提出

目次

第1章　序　論

1.1.　並列処理

１つの目的を持った処理を、いくつかの処理に分割して、これらを同時に行うのが並列処理である。例えば、１００枚ある答案を採点して一番成績が良いものを１つ選ぶのに、採点者が２人いれば、１人ですべてを採点するときの半分の時間で終わるし、３人いれば３分の１の時間で終わることは簡単に想像できる。しかし、ここで１００人いたら１００分の１になるだろうか。答案用紙を配ったり、集めたり、一番点数を探すのにも時間がかかる。計算機で並列処理を行う場合においても、同じような問題が起こる。このように、処理を分割したために新たに必要となる（本来は必要のない）処理をオーバーヘッドと言う。２人や３人で処理をするときにはほとんど問題にはならないが、５０人や１００人となるとそのオーバーヘッドは目立ってくる。これは、全体の作業量と作業者の数のバランスの問題である。計算機の世界でいうと、作業量とは処理すべきデータ量（問題規模）で、作業者とはプロセッサである。

1.2.　並列処理の目的

並列処理の目的は大きく分けて 2 つある。第一の目的は、問題をより小さいサイズの部分問題に分割し、複数のプロセッサ上で各部分問題を同時に処理することにより処理速度を高速化することである。第二の目的は、耐故障(Fault Tolerant)性や無待機(Wait Free)性を得ることである。システムを多重化することにより、いくつかのプロセッサが故障してもシステム全体としては処理を行なうことができる。本研究では、第一の目的である処理の高速化を取り上げる。

1.3.　並列計算機 [1)]

1980 年代後半、並列計算機の製品化が活発になり，現在ではスーパーコンピュータや大規模サーバなどはすべて並列型になっている。ワークステーションやパソコンも，高性能のものは2～4個のプロセッサをもっている。また最近は，複数のコンピュータをネットワーク接続して並列／分散処理をさせる「コンピュータクラスタ」も利用されている。さらには，地球規模でネットワーク接続されたコンピュータで分散処理を行う「グローバルコンピューティング」も研究されている。並列計算機は、共有メモリ型並列計算機(Shared Memory Parallel Computer)で分散メモリ型並列計算機(Distributed Memory Parallel Computer)の２つに分類される。共有メモリ型計算機は共有メモリ(Shared Memory)とそれに接続した複数のプロセッサから成る。一方、分散メモリ型並列計算機は局所メモリ(Local Memory)をもつ複数のプロセッサと、それらを結びつけるネットワークから成る。

一般に共有メモリ型計算機は通信遅延が短く、プロセッサ間の同期も取り易

いが、プロセッサ数を増やすことは困難である。逆に分散メモリ型計算機は比較的多数のプロセッサを持つことができるが、プロセッサ間の通信には時間がかかる。

このため、プロセッサ数が少ない並列計算機は共有メモリ型が、プロセッサ数が多い並列計算機は分散メモリ型が主流になっている。

1.4. 並列アルゴリズムと並列計算モデル

並列計算機上では、従来の逐次アルゴリズムを元にしたプログラムは効率良く実行できず、並列処理を考慮した並列プログラムを作る必要がある。並列プログラムの元となるアルゴリズムが並列アルゴリズムである。並列アルゴリズムは仕事をどのように分割し、各プロセッサに対して分割した仕事を割り当てるかを考慮して設計される。

一般的に、並列アルゴリズムの設計・開発は並列計算機を抽象化した並列計算モデルの上で行われる。並列計算モデルを用いることにより、個々の並列計算機ごとに異なる性質に囚われることなく並列アルゴリズムの設計・開発を行うことができる。

逐次計算では、標準的な計算モデルとして RAM(Random Access Machine)が用いられる。一方、並列計算ではメモリの形式、プロセッサ間の通信や同期、データの扱い方の違いなどが異なるため標準的な計算モデルを決めにくい。このため PRAM(Parallel Random Access Machine),メッシュ接続型並列計算モデル、ハイパーキューブ接続型並列計算モデルなど様々な並列計算モデルが提案されている。

1.5. 本研究の目的

本研究では、分散メモリ型並列計算モデルである BSP(Bulk-Synchronous Parallel)モデル上で高速な整列アルゴリズムを提案することを目的とする。整列問題は様々な分野に応用できる基本的な問題であり需要が高い。このため並列計算機上で高速に実行できるアルゴリズムを開発することは非常に重要である。また、BSP モデルは通信遅延や同期時間等のコストを考慮して並列アルゴリズムを設計することができ、多くの実在する並列計算機に対応する汎用性高いモデルであるとして近年注目されているモデルである。

第2章　準　備

2.1.　並列アルゴリズム

アルゴリズム(Algorithm)とは、与えられた問題を解決するための論理や手順のことである。この手順はどのような処理をどのような順番で行なうか、曖昧性が無いように記述せねばならない。

並列アルゴリズム(Parallel Algorithm)は、並列計算モデル上で実行させるためのアルゴリズムである。並列アルゴリズムは、与えられた問題をどのようによりサイズの小さい部分問題に分割し、それをどのように各プロセッサに割り当てるかを記述せねばならない。そのため、一般的には逐次アルゴリズムをそのまま並列計算モデル上で動かすことはできず、並列計算モデル用に新たにアルゴリズムを作成する必要がある。

2.2.　PRAM (Parallel Random Access Machine)[2)]

共有メモリ(Shared Memory)とそれに接続された複数のプロセッサから成る並列計算機を共有メモリ型並列計算機(Shared Memory Parallel Computer)と呼ぶ。PRAM(Parallel Random Access Machine)は共有メモリ型並列計算機を抽象化したモデルである。**図 1** に PRAM の概念図を示す。

並列アルゴリズムの実行中、各プロセッサは入力データの読み出し、中間結果の読み出し及び書き込み、最終結果の書き出しのため共有メモリにアクセスする。PRAM は、各プロセッサは共有メモリ上の任意の位置にあるメモリセルに対して、1 単位時間でデータの読み書きができ、また全ての演算は 1 単位時間でできると仮定されている。また 1 単位時間に全てのプロセッサで同期が取られる完全同期型モデルである。

共有メモリ型であるため、PRAM ではデータの局所性は考慮されない。また、プロセッサ間の通信は共有メモリを通して行なわれること、完全同期型であることから、プロセッサ間通信や同期にかかる遅延は無視されている。

2.3.　BSP (Bulk-Synchronous Parallel)[3)]

BSP(Bulk-Synchronous Parallel)とは Valiant により提案された分散メモリ型並列計算モデルである。ＢＳＰは以下の要素からなる。

- 局所メモリをもつ複数の各プロセッサは１～Ｐの意識別番号を持ち、Ｐ1、Ｐ2、Ｐ3・・・$Ｐ_p$と表される。
- プロセッサ間の１対１メッセージ通信を行う完全結合網
- プロセッサ間のバリア同期を実現するための同期機構

ＢＳＰの概念図を**図 2** に示す。ＢＳＰモデルは、通信遅延、同期等のコストを表すために以下の３つのパラメタを持つ。

- Ｐ：プロセッサ数
- ｇ：１個のメッセージを送信あるいは受信するためにかかる時間。
- Ｌ：バリア同期時間。これは同時に通信遅延時間を表すパラメタでもある。

ＢＳＰ上モデル上での並列アルゴリズムは、各プロセッサが実行するプログラムにより表される。各プロセッサが実行するプログラムはスーパーステップ

の列からなる。各スーパーステップは内部計算命令の列からなる内部計算フェーズと、送信命令、受信命令の列からなる通信フェーズで構成されており、各プロセッサはスーパーステップの命令を非同期に実行する。また、スーパーステップの命令を終了後、プロセッサ間でバリア同期を取り、次のスーパーステップの実行に移る。メッセージの受信については、各スーパーステップ中の通信フェーズで送信されたメッセージは同一のスーパーステップの通信で受信されるが、そのメッセージはその次のスーパーステップ以降でしか利用できないものと仮定する。あるスーパーステップで各プロセッサP_i($1 \leqq i \leqq P$)がそれぞれw_i個の内部計算命令とh_i個の送信命令あるいは受信命令を実行するとき、そのスーパーステップ全体の実行時間tは

$$t = O(w + g h + L) \cdots\cdots\cdots (1)$$

であると仮定されている。ただし、$w = \max\{w_1, w_2, \cdots w_p\}$、$h = \max\{h_1, h_2, \cdots h_p\}$である。

スーパーステップの特性を持つことにより特性を保たれる。

- データ通信の依存関係の解析ができる。
- バリア同期に必要な時間を通信遅延ととらえることにより、通信コストを考慮できる。
- バリア同期以外の同期はないという非常に緩い同期を仮定するだけでいい。

2.4. 並列クイックソート(Parallel quick sort)[4]

クイックソートはランダムなデータに対してはもっとも効率的なソート法である。クイックソートの基本的な流れを説明する。データの中からある値を選び、それを基準に全データを振り分けて並べ直し、結果的にその基準値を小さいものは前に、大きいものは後ろにくるように前後に分割する。同じ作業を、前半および後半についても行う。さらに、この分割作業を、すべての区間について行う。すべての細分化された区間に含まれるデータ数が１つ以下になると、ソートされた状態になる。並列クイックソートは分割された区間ごとにプロセッサを割り当てることにより、高速化を目指した並列アルゴリズムである。

2.5. 並列クイックソートの手順

ｎ個のデータが配列に入っているとする。そして、データを分ける基準となる値のことを、基準値と呼ぶ。ＲＡＭ上で逐次クイックソートを行う場合、その手順は次のようになる。

（１） ソートするデータ区間[ｌ, ｒ]

（２） 区間内のデータ数により次のことをする。

① データ数が１以下のとき、何もしない。

② データ数が２のとき、逆順なら入れ替える。

③ データ数が３以上のとき、次のようにデータを振り分ける。

（ａ）区間内から基準値を１つ選ぶ。

（ｂ）区間内のデータに対し、基準値より小さいものと大きいものとを２つの区間[ｌ,ｍ－1]と[m, ｒ]にそれぞれ振り分ける。

（ｃ）両方の区間[ｌ,ｍ－1]および[m, ｒ]に対し、データの振り分けを再帰

的に行う。
複数のプロセッサを用いて並列処理を行う場合、(３)③(ｃ)の区間[ｌ,ｍ－1]および[m, ｒ]の処理は２個のプロセッサで並列に行うことができる。従って、並列処理を用いてクイックソートを行う場合、区間の個数がＰ個になるまでは、区間を分割するごとに１つの区間に１台のプロセッサを割り当てていく。区間の個数がＰ個になった後は、各プロセッサで割り当てられた区間を逐次クイックソートを用いて整列すればよい。ＰＲＡＭ上での並列クイックソートの概念図を第３図に示す。図において同じ色で示されているデータは同じプロセッサがする。

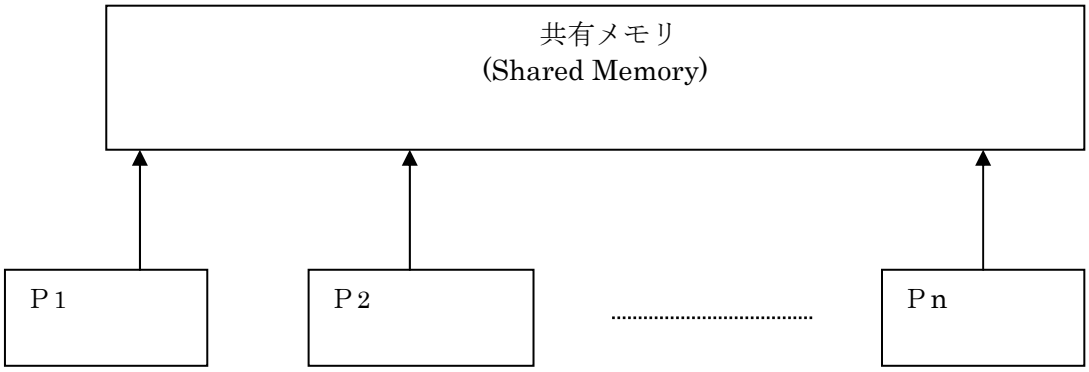

図 1 並列ランダムアクセスマシン
Fig.1 A Parallel random access machine

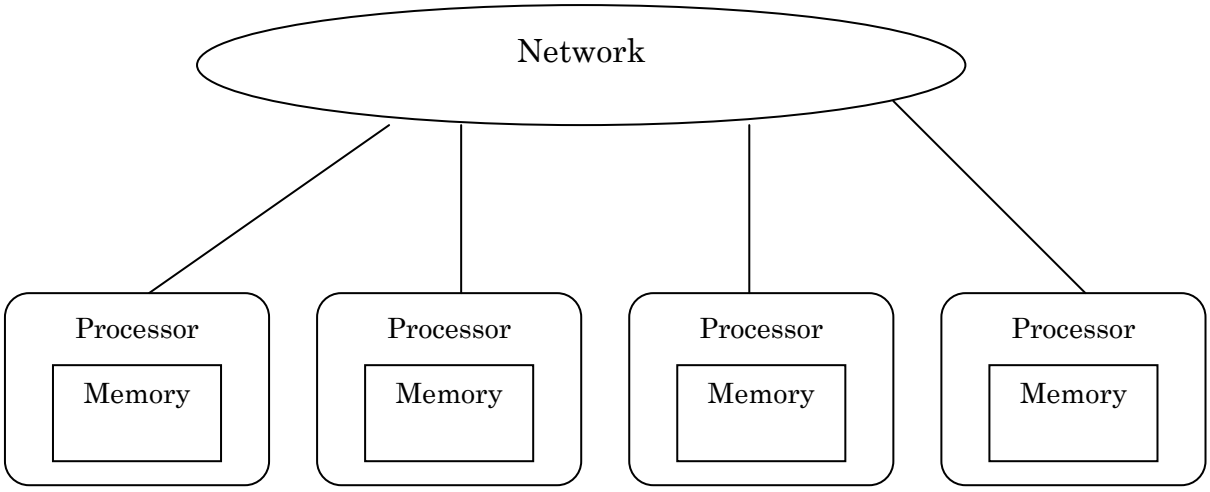

図 2 BSP モデル
Fig.2 The BSP Model

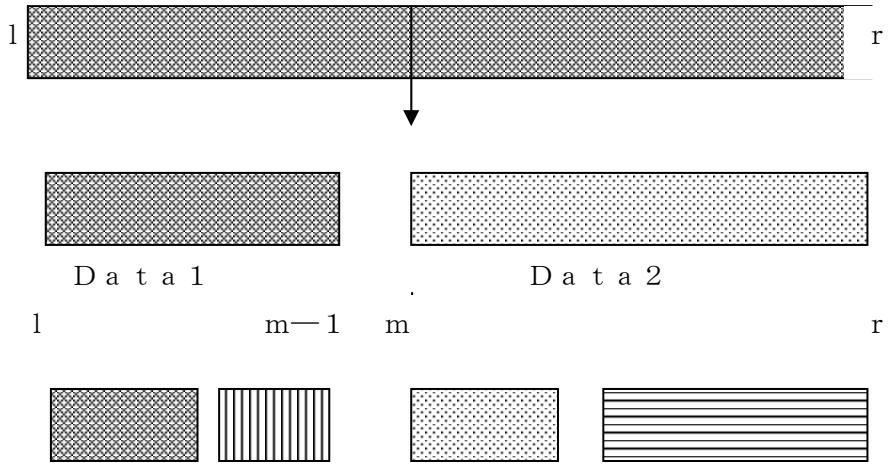

図 3 PRAM 上での並列クィックソート

Fig. 3　Parallel quick sort on PRAM

2.6.　BSP 上でのクイックソート

逐次クイックソートでは区間[ｌ, ｒ]のデータを２つの区間[ｌ,ｍ－1]と[m, ｒ]に分割するとき、[ｌ,ｍ－1]と[m, ｒ]内のデータの個数が等しくなるとき最も効率が良くなる。すなわち、区間[ｌ, ｒ]内のデータの個数がｍ個のとき、区間[ｌ, ｍ－1]と[m, ｒ]内のデータの個数がｎ／２となるように、[ｌ, ｒ]内のデータの中央値を基準値として用いればよい。ＰＲＡＭ上での並列クイックソートも同様に、ｎ個のデータから減る区間を[ｌ, ｒ]はそれぞれｎ／２個のデータから成る区間[ｌ,ｍ－1]および[m, ｒ]に分割するのが効率的である。しかし、ＢＳＰモデル上ではプロセッサＰ1 が区間[ｌ, ｒ]を区間[ｌ,ｍ－1]および[m, ｒ]に分割した後、プロセッサＰ2 で区間[m, ｒ]を処理するためには、区間[m, ｒ]のデータをＰ1 からＰ2 へ送信しなければならない。ＢＳＰの仮定より、ｎ／２個のデータを送信する通信時間ｔは

$$t = g\,n/2 + L \quad \cdots\cdots (2)$$

である。一方、区間[ｌ,ｍ－1]はＰ1 が引き続き処理を行うため通信時間がかからない。このため、[ｌ,ｍ－1]と[m, ｒ]の処理時間に差ができるため、効率の低下を招く、従って、ＢＳＰモデルでは基準値を中央値ではなく、通信遅延を考慮して選択せねばならない。つまり、区間[ｌ,ｍ－1]を処理するための内部計算時間＝区間[m, ｒ]を処理するための内部計算時間+区間[m, ｒ]のデータの通信時間となるように基準値を選択する。ＢＳＰ上での並列クイックソートの概念図を図 4 に示す。

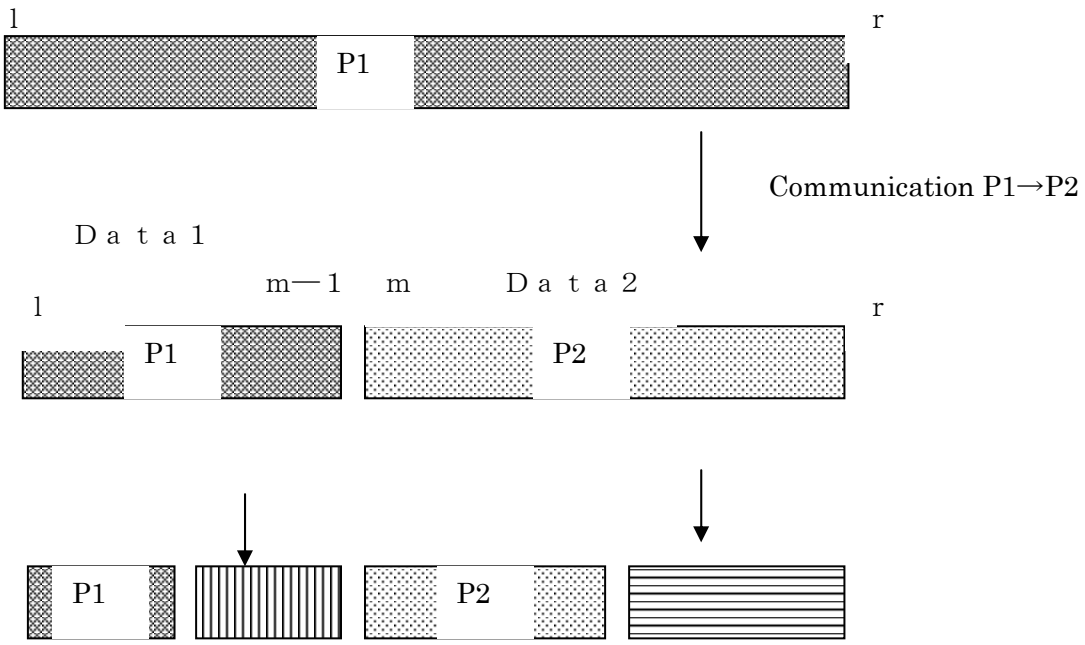

図 4 BSP 上での並列クィックソート

Fig.4　Parallel quick sort on BSP

第3章　方　法

3.1.　ＢＳＰ上でのクイックソートアルゴリズム

本研究ではＣ言語を用い、ＢＳＰ上でのクイックソートアルゴリズムのプログラムを作成した。本研究で作成したプログラムは、データ数ｎ、プロセッサ数Ｐ、通信帯域幅ｇ、同期時間Ｌが与えられたとき、中央値を基準値として用いた場合の実行時間と基準値を最適な位置に動かしたときの実行時間とその位置を出力するプログラムである。

3.2.　実行時間の測定

データ数Ｎ＝１０００(個)、プロセッサ数ｐ＝１００(台)とし、通信遅延と帯域幅をそれぞれ固定した場合の基準値の取り方を変化させ、実行時間および基準値の位置を測定した。

3.2.1.　通信遅延を固定したときの実行時間の測定

通信遅延ｌ＝１６とし、基準値の取り方を変化させたときの帯域幅 g と実行時間ｔの関係を測定する。このとき、基準値は最適値、中央値の両方について測定する。

3.2.2.　帯域幅を固定したときの実行時間の測定

帯域幅ｇ＝１６とし、基準値の取り方を変化させたときの通信遅延ｌと実行時間ｔの関係を測定する。このとき、基準値は最適値、中央値の両方について測定する。

3.3.　プログラムの説明

付録に本実験で作成したプログラムを示す。
このプログラムは 3.2.2. の実験で使用したプログラムである。このプログラムで帯域幅を固定することにより、3.2.1. の実験を行った。

第4章　結　果

4.1.　通信遅延を固定したときの実行時間の測定

データ数 N＝１０００(個)、プロセッサ数 p ＝１００(台)、 通信遅延 l ＝１６とし、基準値の取り方を変化させたときの帯域幅 g と実行時間 t の関係を測定した。これを表 1 と図 5 に示す。ここで分離位置(Separation position)とは中央値からどれだけ離れているかを指す。

4.2.　帯域幅を固定したときの実行時間の測定

データ数 N＝１０００(個)、プロセッサ数 p ＝１００(台)、帯域幅 g ＝１６とし、基準値の取り方を変化させたときの通信遅延 l と実行時間 t の関係を測定した。これを表 2 とエラー! 参照元が見つかりません。に示す。

表 1 帯域幅 g と実行時間 t の関係

Table.1 Relation of Bandwidth g and Execute time t

Bandwidth g	Execute time (Pivot= median)	Execute time (Pivot=optimal)	Separation position
1	12658	12658	5
2	12728	12329	6
4	14870	13860	6
8	17872	16397	6
16	27388	18354	7
32	36059	22443	7
64	52934	24671	8

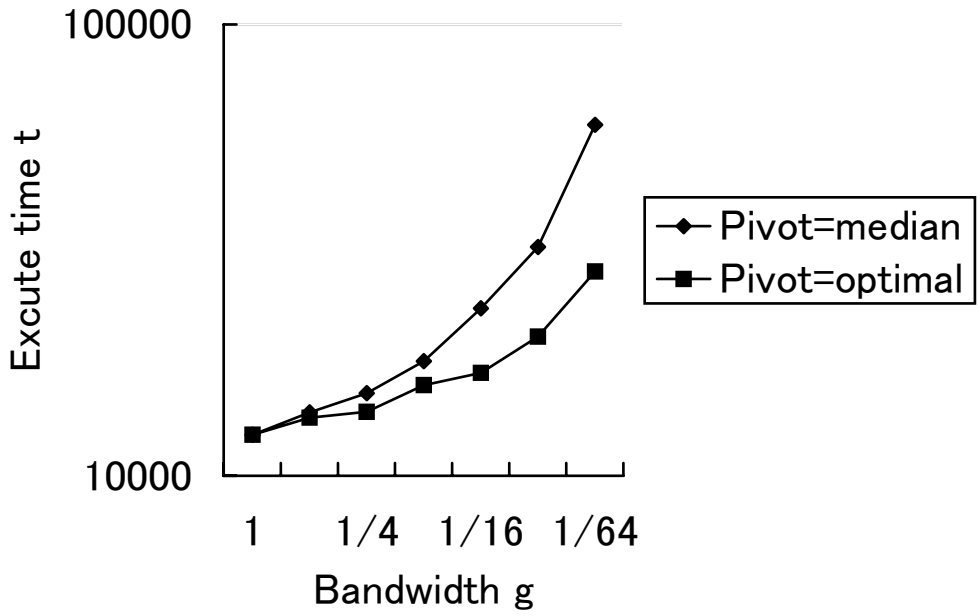

図 5 帯域幅 g と実行時間 t の関係

Fig.5 Relation of Bandwidth g and Execute time t

表 2 通信遅延 l と実行時間 t の関係

Table.2 Relation of Latency l and execute time t

Latency l	Execute time (Pivot= median)	Execute time (Pivot= optimal)	Separation position
1	24107	18223	7
2	23321	18030	7
4	26636	18020	6
8	23953	18774	6
16	24363	18492	7
32	25300	18204	7
64	24043	16333	7

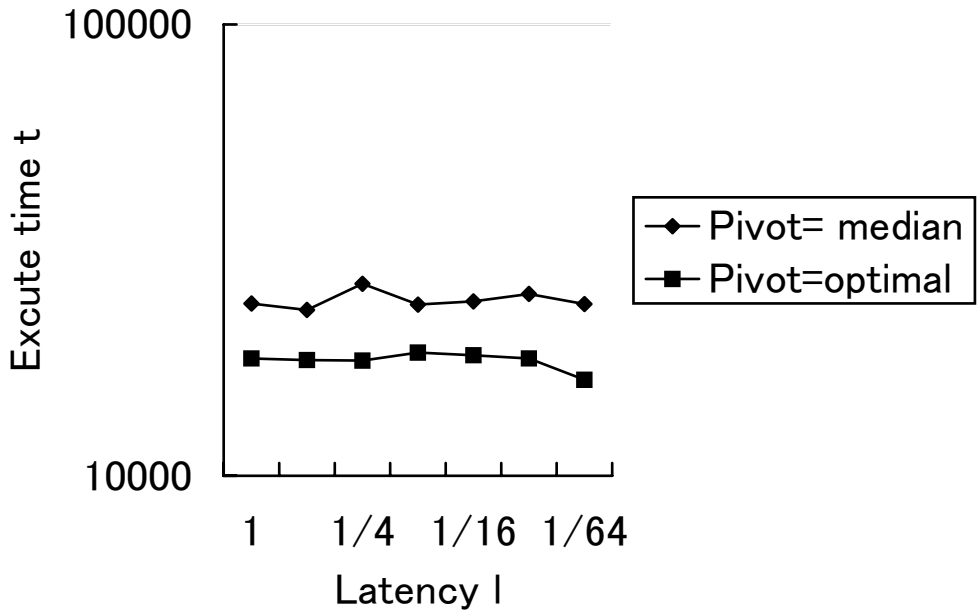

図 6 通信遅延 l と実行時間 t の関係

Fig.6 Relation of Latency l and execute time t

第5章　考　察

5.1.　通信遅延 l を固定したときの実行時間

図 3 から基準値を最適値としたほうが実行時間が短縮できることがわかる。これは、ＢＳＰ上では実行時間は内部計算時間と通信時間の和であることから、通信時間と内部計算時間のバランスが良くなったためであると考えられる。よって、基準値の選び方を工夫することにより、実行時間の短縮が可能であることが明らかとなった。表 1 から帯域幅 g が小さくなるにつれ、最適値も中央値から離れていくことがわかる。

5.2.　帯域幅 g を固定したときの実行時間

図 4 から 4.1. と同様に基準値を最適値としたほうが実行時間が短縮できることがわかる。しかし、帯域幅 g の変化によって実行時間が大きく変化しないことから、実行時間をさらに短縮するためには通信遅延 l を変化させる必要があると考えられる。表 2 において分離位置はほとんど変化していない。最適値の選び方を考慮する際、帯域幅 g には依存しないことが考えられる。

5.3.　プロセッサの実行時間

図 3 と図 4 からプロセッサの実行時間の変化の仕方が異なることがわかる。図 3 では帯域幅 g が小さくなるにつれ、実行時間も小さくなる。図 4 では通信遅延 l が小さくなっても実行時間に大きな変化はみられない。以上のことから最適な基準値の位置は通信帯域幅の大きさによる影響が大きく、一方、通信遅延時間にはほとんど影響されないことがわかる。

5.4.　今後の課題

帯域幅 g と通信時間 l の関係に応じた最適値を求めることが今後の課題である。

第6章　結　論

本研究により、ＢＳＰ上のアルゴリズムにおいて通信遅延時間ｌ、帯域幅ｇとの実行時間が明らかとなった。また通信遅延ｌと帯域幅ｇが基準値を取るときに与える影響がわかった。これらを具体的に以下に示す。

- データ数N、プロセッサ数Ｐを一定としたとき、実行時間は帯域幅が小さくなるにつれ実行時間も少なくなる。このとき、最適値は中央値から離れていく。
- 帯域幅ｇ＝１／６４のとき、中央値を基準とする従来の方法に比べ、最適値を用いた場合約２倍のスピードアップ率を達成した。
- データ数N、プロセッサ数Ｐを一定としたとき、通信遅延は実行時間にあまり影響を及ぼさない。このとき、通信遅延が変化しても最適値は中央値をとらないが、分離位置もあまり変化しない。
- ソーティングの基準値に最適な値をとることが高速な計算をするために必要である。

謝辞

本研究の実験、報告書の作成するにあたり石水先生に多大なお世話になり、誠に感謝申し上げます。また、研究場所が今尾先生の制御工学研究室ということもあり、同研究室の方々にもご迷惑をおかけいたしまして申し訳ございませんでした。改めて、今尾先生を含む制御工学研究室の方々にも感謝の気持ちを申し上げます。

参考文献

1) 渋沢　進：“並列分散処理入門”、培風館、東京、４、（１９９８）
2) J. JáJá. : “An Intoroduction to Parallel Algorithms.”、Addison Wesley Publishing Company、（１９９２）
3) L.G. Valiant : “A　Bridging Model for Parallel Computation,”Comm. of the ACM、（１９９０）
4) 浪平博人 : “データ構造とアルゴリズム３　ソート・検索”、ＣＱ出版社、東京、３７～４９、（１９９５）