卒業研究報告書

題 目

BSP モデル上のソーティングアルゴリズム

指導教員

石水隆助手

報告者

01-1-26-003

松川 怜

近畿大学理工学部電気工学科

平成 1 7 年 2 月 19 日提出

目次

第 1 章　　序　論 - 1 -
1.1.　並列コンピュータと並列アルゴリズム - 1 -
1.1.1.　アルゴリズムとは何か - 1 -
1.1.2.　並列コンピュータ [1] - 1 -
1.1.3.　並列アルゴリズム（Parallel Algorithm）[2] - 2 -
1.1.4.　並列計算モデル - 2 -
1.2.　本研究の目的 - 3 -
第 2 章　　準　備 - 4 -
2.2.　BSP モデル [4] [5] - 5 -
2.3.　並列クイックソート - 8 -
2.4.　BSP 上でのクイックソート - 10 -
第 3 章　方　法 - 11 -
3.1.　BSP 上でのクイックソートアルゴリズム - 11 -
3.2.　基準値の選び方 - 11 -
3.2.1.　最適な基準値 - 11 -
3.2.2.　サンプルを用いた基準値の選び方 - 11 -
3.2.3.　シミュレートプログラム - 12 -
第 4 章　結　果 - 13 -
第 5 章　考　察 - 14 -
第 6 章　結　論 - 15 -
付　録 - 18 -

第1章　序　論

1.1. 並列コンピュータと並列アルゴリズム

1.1.1. アルゴリズムとは何か

アルゴリズム（Algorithm）は、問題を解くための手段を定めたものである。この手順は、どのような操作をどのような順序で行うかを、曖昧な点の残らないようにきちんと定めたものでなければならない。

手順を明確に定めてあれば、計算機をその手順どおり動かして問題を解かせることができる。アルゴリズムという概念自身は計算機と無関係に成立するが、普通は計算機に問題を解かせるための手順を指す。

1.1.2.　並列コンピュータ [1)]

並列コンピュータ（Parallel Computer）とは、ネットワークあるいは共有メモリによって結ばれた複数個のプロセッサ（CPU）を同時に実行させて、ある問題に対する計算処理の高速化を達成することを目標とするコンピュータシステムを指している。ある問題を部分問題に分割して複数個のプロセッサの個々（要素プロセッサという）に分担させ、それを行うコンピュータシステムを並列処理システムという。

並列コンピュータは並列処理による問題の高速処理を主目標としており、その時代の最高性能のコンピュータの座をベクトルコンピュータと争ってきているスーパーコンピュータである。

並列コンピュータの歴史は、VLSI 技術の発達によって実現されている要素プロセッサ個数の増大過程でもある。数大規模のミニコンなどをチャネル結合した時代を源として、マイクロプロセッサの普及とともに、並列コンピュータとして集積できる要素プロセッサ個数も飛躍的に増加している。現在では、10^4～10^5 個要素プロセッサ規模の超並列コンピュータも出現している。

並列コンピュータは、共有メモリ型並列コンピュータ（Shared Memory Parallel Computer）と分散メモリ型並列コンピュータ（Distributed Memory Parallel Computer）に分けられる。

共有メモリ型並列コンピュータは、共有メモリ（Shared Memory）とそれ接続した複数のプロセッサから成る。一方、分散メモリ型配列計算機は、局所メ

モリ（Local Memory）を持つ複数のプロセッサとそれらを結び付けるネットワークから成る。

一般的に共有メモリ型並列計算機は、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、分散メモリ型並列計算機は、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では共有メモリ型が、プロセッサ数が多い並列計算機では分散メモリ型が主流となっている。

1.1.3. 並列アルゴリズム（Parallel Algorithm）[2)]

並列計算機上では、対照となる問題をよりサイズの小さい部分問題に分割し、各プロセッサがそれぞれに部分問題を同時にすることにより高速化を行える。

p 台のプロセッサを使えば、最大 p 倍の高速化が可能な理屈であるが、実際にそこまで高速化するのは難しい。それは p 台のプロセッサに仕事を均等に割り振るのが困難だからである。ある問題は、別の問題の計算が終わってからでないと始められないことがある。この二つの問題の計算を別のプロセッサに割り振った場合、一方の計算が終わるまで、もう一方は待っていなければならない。下手をすると無駄な待ち時間ばかり多くなって、1 台で計算したのと大差ないということになりかねない。このようなことがないようにアルゴリズムを設計するのは、普通のアルゴリズムの設計とは違った難しい問題である。また、プロセッサの間でデータをやりとりするには,単に計算をするのに比べて時間がかかるのが普通であり、通信の量を減らすことも重要な評価のポイントとなる。

並列アルゴリズムの場合、どのような計算機構（アーキテクチャ）の上で実現するかということが実際的な問題になる。プロセッサが 1 台の場合、現実の計算機はどれも大差ない。これに対して、複数のプロセッサを結合した計算機構には様々な種類がある。特に、プロセッサ同士がどのような手段で連絡を取り合うかが問題である。

並列計算機は共有メモリ型並列計算機と分散メモリ型並列計算機の二つに分けられるため、それぞれの種類ごとに色々なアルゴリズムが工夫されている。

1.1.4. 並列計算モデル

並列アルゴリズムの設計・解析は、並列計算機を抽象化した並列計算モデル(Parallel Computing Model)上で行われる。

並列計算モデルは、メモリの形式、ネットワークの形状、通信遅延の特性などが異なる様々なモデルがある。本研究では、代表的な並列計算モデルである

PRAM（Parallel Random-Access Machine）及び、近年注目されている並列計算モデル BSP（Bulk-Synchronous Parallel）モデルを扱う。BSP モデルは、Valiant により提案された並列計算モデルであり、最近の並列計算において重要とされている通信コストを、通信遅延 L、通信路帯域幅の逆数 g といったパラメタにより表すことを可能にしたモデルである。

1.2. 本研究の目的

本研究は BSP モデル上で高速な整列アルゴリズムを作ることを目的とする。整列は様々な分野で利用される重要な基本問題の一つであり、BSP モデル上においても高速な整列アルゴリズムが必要とされている。

第2章　準　備

2.1.　共有メモリ型並列計算モデル [3)]

図 1 のように、複数のプロセッサがメモリを共有したコンピュータを共有メモリコンピュータ（Shared Memory Computer）と呼ぶ。代表的な共有メモリコンピュータによる並列計算モデルとして、並列ランダムアクセスマシン（PRAM，Parallel Random-Access Machine）がある。

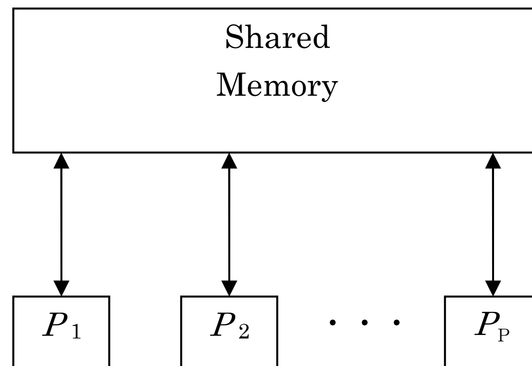


図 1 共有メモリコンピュータ

Fig.1. Shared Memory Computer

PRAM の各プロセッサは共有メモリ（SM，Shared Memory）上の任意の位置にあるメモリセルに対して一単位時間で読み書きでき、また全ての演算は一単位時間で行うことができると仮定されている。また、PRAM は細粒度同期式であり、一単位時間ごとに全てのプロセッサで同期が取られる。プロセッサ間の通信は共有メモリを通じて行われる。

並列アルゴリズムの実行中、P 個のプロセッサは，入力データを読んだり、中間結果を読んだり書いたり、最終結果を書いたりするために、共有メモリにアクセスする。複数のプロセッサが共有メモリにアクセスする場合にそれが全て異なる位置にあるメモリセルに対してであれば自由に読み書きできる。一方、複数のプロセッサが同じ位置にあるメモリセルに対してアクセスする場合、そ

れをどう処理するかにより PRAM は以下の 4 種類に分類される。

1. 排他読み出し排他書き込み（EREW，Exclusive-Read Exclusive-Write）
2. 同時読み出し排他書き込み（CREW，Concurrent-Read Exclusive-Write）
3. 排他読み出し同時書き込み（ERCW，Exclusive-Read Concurrent-Write）
4. 同時読み出し同時書き込み（CRCW，Concurrent-Read Concurrent-Write）

EREW PRAM は、あるアドレスへのアクセスが一度に 1 プロセッサに限られるので、PRAM の 4 つの部分クラスの中で最も計算能力が低い。EREW PRAM に対するアルゴリズムは、同じメモリ位置から読み出したり書き込んだりするプロセッサ数を、一度に高々1 個に限定して実行されるが、アルゴリズムの実行に必要な計算時間またはメモリ容量をいくらか増やすことによって、多重アクセスをシミュレートすることができる。

2.2. BSP モデル [4) 5)]

BSP（Bulk-Synchronous Parallel）モデルは Valiant によって提案された非同期式並列計算モデルであり、以下の構成要素から成る。

- 局所メモリを持つ複数のプロセッサ
- 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
- プロセッサ間の同期を実現するための同期機構

図 2 に BSP モデルを図表化したものを示す。BSP モデルは、各プロセッサが局所メモリを持ち、それぞれネットワークに繋がっている。各プロセッサはネットワークを通して 1 対 1 通信を行うことができる。また、同期機構を持ちすべてのプロセッサでバリア同期を取ることができる。ここで、バリア同期とは、協調して動作する多数のプロセッサの歩調を合わせることを目的とした同期プリミティブである。バリア同期を実行して同期を取る場合、全てのプロセッサがバリアに到達するまでどのプロセッサも実行を継続できず、封鎖される。

BSP のネットワーク及び同期機構の特性は以下のパラメタによって抽象化さ

れている。

- L : 通信遅延時間かつバリア同期時間
- g（≦L）: 1 個の送信命令または受信命令の実行に必要な時間
- P : プロセッサ数

BSP 上での命令の実行は以下のように仮定されている。

- 各プロセッサは 1 単位時間に 1 内部計算命令を局所メモリにのみ基づいて実行する。
- メッセージ 1 個の送信命令または受信命令の実行は g 単位時間で行われる。ただし、1 メッセージは 1 語からなるものとし、サイズ 1 のメッセージと呼ぶ。

BSP 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。図 3 に BSP モデル上での並列アルゴリズムの実行の概念図を示す。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と仮定する。

あるスーパーステップにおいて、全てのプロセッサで命令の実行を終了してから L 時間以内にバリア同期が取られ、次のスーパーステップの実行に移る。よって、あるスーパーステップにおいて、各プロセッサが高々 w 個の内部計算命令、高々 h 個の送信命令または受信命令を割り当てられた場合、そのスーパーステップの実行には $O(w+gh+L)$ 時間かかる。

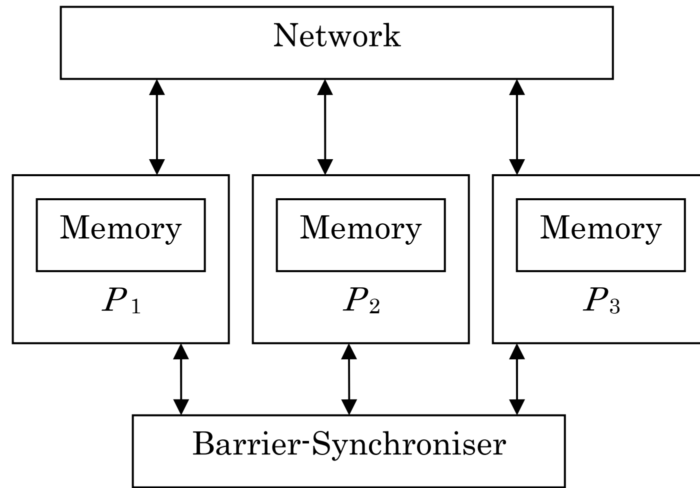


図 2 BSP モデル

Fig.2. Bulk-Synchronous Parallel Model

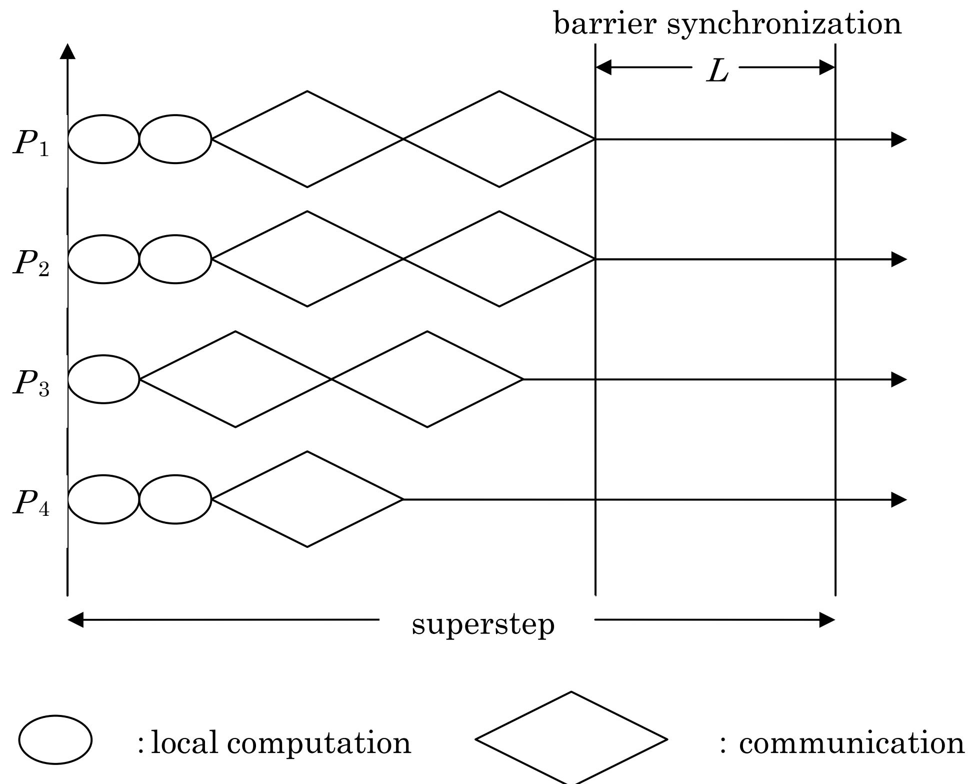


図 3 BSP モデル上での実行の概念図

Fig.3. Execution on the Bulk-Synchronous Parallel Model

2.3. 並列クイックソート

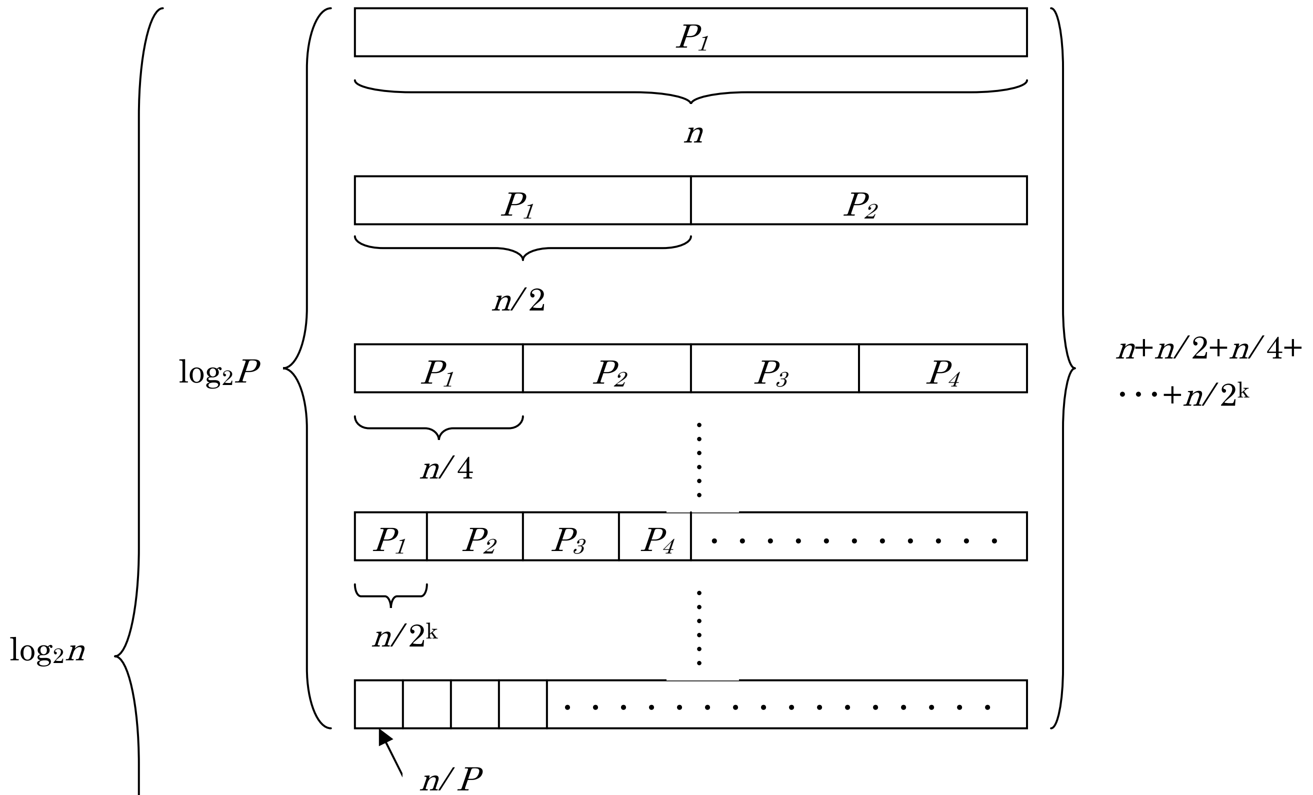
n 個のデータが与えられた時、それを昇順に並べる操作がソートである。ソートは多くの問題に利用される基本的な問題であり、BSP を初め、様々なモデル上で高速なアルゴリズムが求められている。RAM 上では高速な整列アルゴリズムとしてクイックソートがまた、PRAM 上では並列クイックソートが提案されている。そこで本研究では、PRAM 上での並列クイックソートをベースとし、BSP 上で高速に整列を行うことができるアルゴリズムを提案する。

図 4 にデータ数 n、プロセッサ数 p の時の PRAM 上での並列クイックソートの実行の様子を示す。

クイックソートは適当な基準値を選び出しデータをその基準値より大きいものと小さいものとの二つに分割する。という操作を繰り返し行うことで整列を行うアルゴリズムである。適当な基準値を選択すれば、充分高い確率で分割後の 2 つの区間の大きさは、分割前のほぼ 1/2 になる。従って、以下では各分割において、区間の大きさは分割前の 1/2 になると仮定する。

まず、プロセッサ P_1 を用いて大きさ n のデータを 2 個の大きさ $n/2$ のデータに分割する。次にプロセッサ P_1、P_2 を用いて大きさ $n/2$ の各データをそれぞれ大きさ $n/4$ のデータに分割する。以下同様に各データについて 1 台のプロセッサを割り当て、それぞれ大きさが半分のデータに分割する。という操作をデータ数が p 個になるまで繰り返す。データ数が p 個になれば各データに 1 台のプロセッサを割り当て、各プロセッサが逐次にクイックソートを行う。

分割区間数 2^k が P より大きい場合



分割区間数 2^k が P より大きい場合

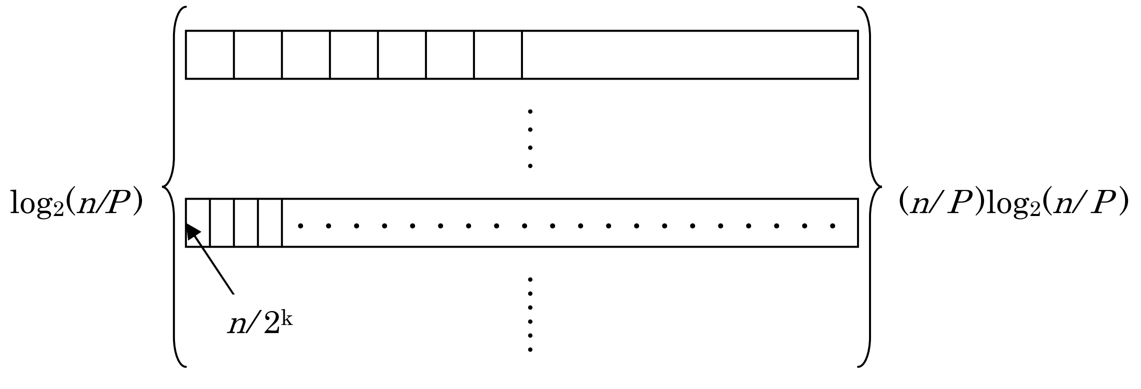


図 4 並列クイックソートの計算時間

Fig.4. Computation time of Parallel Quick Sort

2.4. BSP 上でのクイックソート

2.3 で示した PRAM 上での並列クイックソートを BSP 上で動かした場合、大きさ n のデータをプロセッサ P_1 を用いてサイズ $n/2$ の 2 個のデータ d_1、d_2 に分割した後、データの片方をプロセッサ P_1 からプロセッサ P_2 に送信しなければならない。仮定より、サイズ $n/2$ のデータの送信にかかる時間は $gn/2+L$ 時間かかる。一方、もう片方のデータはプロセッサ P_1 自身が引き続き処理を行うため通信時間はかからない。このため二つのデータの処理時間にアンバランスが生じ、効率が悪くなり実行時間の低下を招く。BSP モデル上で効率良く並列クイックソートを行うには、データを分割する際に $n/2$ に分割するのではなく、プロセッサ P_1 がデータ d_1 を処理する内部計算時間に対しデータ d_2 を送信する時間とプロセッサ P_2 がデータ d_2 を処理する内部計算時間との和が等しくなるように分割すればよい。そこで第 3 章以下ではその方法について説明する。

第 3 章　方　法

3.1.　BSP 上でのクイックソートアルゴリズム

2.4 で述べた通り PRAM 用のアルゴリズムは通信時間が考慮されておらず、BSP 上で実行させた場合効率良く実行できない。その理由としては、PRAM 上の並列クイックソートではデータを二つに分けるための基準値として中央値を選んでいたが、この手法は BSP 上では通信時間が大きくなり効率良くないからである。従って、BSP 上では通信遅延と通信帯域幅を考慮した基準値を選ぶことが必要になってくる。そこで、本研究では適切な基準値の選び方を検討する。

3.2.　基準値の選び方

3.2.1.　最適な基準値

大きさ n のデータを k:(1-k)という比で分割すると、一方は大きさ kn、もう一方は大きさ(1-k)n というデータになる。(1-k)n 個のデータの送受信にかかる通信時間は $g(1-k)n+L$ 時間である。他のプロセッサに送ったデータはソート後、再び送り返してもらう必要がある。よって、通信にかかる時間は前記の時間の 2 倍の $2\{g(1-k)n+L\}$である。また、kn と(1-k)n 個のデータをソートするのにかかる内部計算時間はそれぞれ $kn\log kn$ と $(1-k)\log(1-k)n$ 時間である。よって、$kn\log kn=2\{g(1-k)n+L\}=(1-k)\log(1-k)n$ という式を k について解けばよい。ここで k は 1 以下の定数であるので $\log kn \leqq \log n$ である。よって、$\log kn$ は $\log n$ と置き換えて計算することができ、従って、解 k は、$k=2(gn+L)/n(\log n+2g)$と求めることができる。

3.2.2.　サンプルを用いた基準値の選び方

この章では基準値を選ぶアルゴリズムについて説明する。n 個のデータの中から kn 番目に大きいデータを探すには一般には選択問題を解けばよい。選択問題は逐次アルゴリズムで $O(n)$時間で解けることが知られている。しかし、選択問題を用いて厳密に k 番目の値を選ぶのは計算量的に効率が悪い。そこで n 個のデータの中から $\log n$ 個のサンプルを取り、$kn\log n$ 番目のデータを選ぶ。確率的に $\log n$ 個中 $k\log n$ 番目のデータは、n 個中 kn 番目のデータの値に近い値になり易いことが知られている。

3.2.3. シミュレートプログラム

本研究では提案したアルゴリズムの正当性を示すために C 言語を用いてシミュレートプログラムを作成した。このプログラムは、入力としてデータ数 N、プロセッサ数 P、通信の帯域幅 G、通信遅延 L が与えられたとき、BSP モデル上で整列を行ったときの実行時間を出力するプログラムである。なお、入力データは 0～M の範囲のランダムな値が設定される。

BSP モデル上で効率良く処理を行うため、クイックソートの分割の基準値は以下のように選んでいる。

1. L,G,P より N 個のデータに対して最適なデータの位置 kN（$0 \leqq k \leqq 1$）を求める。
2. N 個のデータの中から $\log N$ 個のデータを抽出し、作業用配列に格納する。
3. select 関数を用いて作業用配列から $k \log N$ 番目に大きい値を基準値として選ぶ。

基準値が選択できたら以下のようにしてデータの分割を行う。

4. 選択した基準値を元にしてデータを二つに分ける。この時、分割にかかる時間を内部計算時間として計測する。
5. 分割したデータの片方を他のプロセッサに送信する。その際に通信にかかる時間を計測する。
6. 通信時間と内部時間の合計を全体の実行時間として計測する。

以上の 2～6 までを繰り返すことによりソーティングを行う。

第 4 章　結　果

本研究で作成したシミュレートプログラムを使用して、データ数 N=200、プロセッサ数 P =16、通信遅延 L =32 とした時の最適な基準値を用いた場合と中央値を基準値とした場合の通信の帯域幅と実行時間との関係を図 5 に示す。

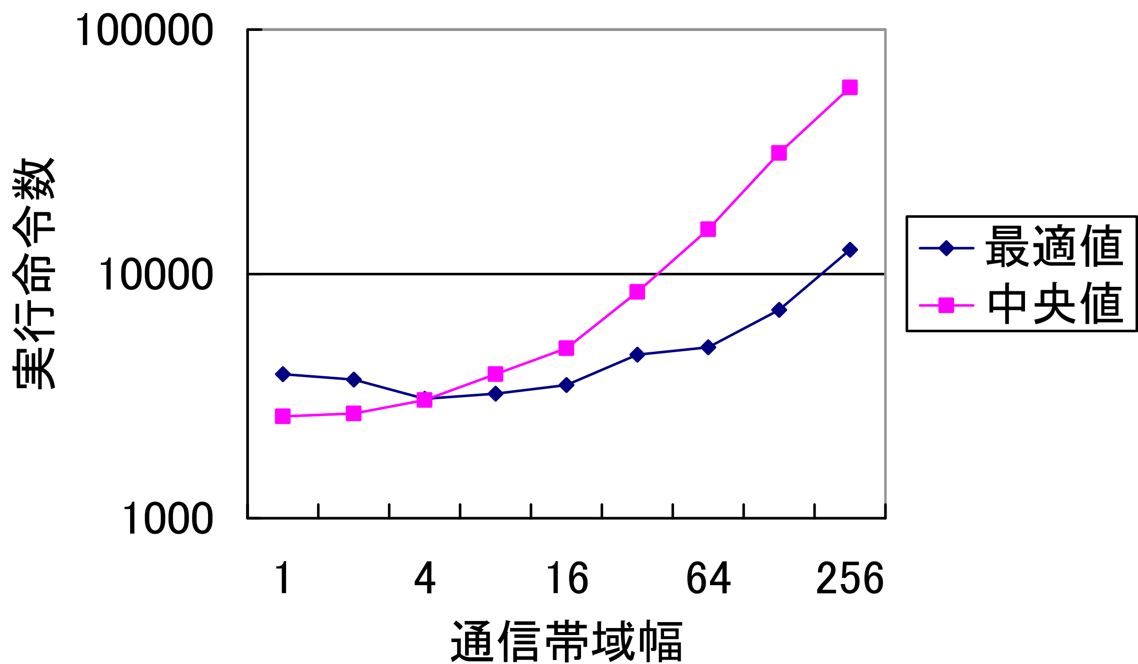


図 5 並列クイックソートの帯域幅特性

Fig.5. Bandwidth characteristic of Parallel Quick Sorting

図 5 より、通信の帯域幅が広い場合は中央値を用いる方が実行時間が短いが、通信の帯域幅が狭くなるにつれ最適値を用いる方が効率が良くなる。また、最適値を用いた場合は通信の帯域幅の減少がある程度の範囲内では実行時間の増加は見られない。一方、中央値を用いた場合では帯域幅の減少に応じて実行時間が増加している。

第 5 章　考　察

最適値を求めるためには中央値を求める場合に比べて余分な内部計算が必要となる。図 5 において、通信の帯域幅が広い場合は中央値を基準値とした場合の方が実行時間が短くなっているが、これは最適値を求めるために余分な内部計算をしたからであると考えられる。逆に、通信の帯域幅が狭い場合は通信時間が大きくなるため、通信時間を減らす工夫をしている最適値の方が効率が良くなる。従って、通信の帯域幅の大きさによって中央値を用いるか最適値を用いるかを切り替えればよい。

第 6 章　結　論

本研究では BSP モデル上でのソーティングアルゴリズムを提案した。また、BSP 上での動作をシミュレートするシミュレートプログラムを作成した。本研究で提案したアルゴリズムは通信帯域幅及び通信遅延に応じて分割の基準値の取り方を変化させている。通信の帯域幅が狭いとき本研究で提案した手法は従来の手法より効率的である。

謝　辞

本研究をするにあたり、C 言語、並列アルゴリズム、クイックソートなどに数え切れないほどのご指導、ご鞭撻を頂いた石水隆先生には感謝の気持ちで一杯です。また、ご迷惑もたくさんおかけしたと思いますが、この一年間本当にありがとうございました。

そして、伊波修一君、北浦邦浩君、北村勇樹君、桜打純視君、柳原大志君には常日頃から助言を賜り、様々な相談にものって頂き、深い感謝、敬愛の気持ちを表します。

参考文献

1) 柴山　潔:“コンピュータアーキテクチャ”、オーム社、東京、365～367（1997）
2) 石畑　清：“アルゴリズムとデータ構造”、岩波書店、東京、417（1989）
3) 渋沢　進：“並列分散処理入門”、培風館、東京、13～15（1998）
4) 石水　隆、藤原　暁宏、井上　美智子、増澤　利光、藤原　秀雄：“選択問題を解く BSP モデル及び BSP*モデル上の並列アルゴリズム”、電子情報通信学会論文誌、D-I Vol.J82-D-I No.4、534～535（1999）
5) L,G.Valiant：“A Bridging Model for Parallel Computation Communication of the ACM、Vol.33、No.8、103～111（1990）

付　録

```
/* Parallel quick sort 並列実行時間（実行目命令数）シミュレート */

#include<stdio.h>
#include<stdlib.h>
#include<time.h>

#define N 200      /* データ数 */
#define M 50       /* 値の範囲 */
#define P 16       /* プロセッサ数 */
#define G 256      /* 通信の帯域幅 */
#define L 512      /* 通信遅延 */

int g,d,n,logn=1;
float k;
int numofpiv;
int b[N];

main() {
  int a[N];  /* 入力データ */
  int t,tmid=0,tmax=0;  /* 実行命令数 */
  int i;
  int select();
  void inputArray(), outputArray();
  int quickSort();
  FILE *fpo;

        srand((unsigned int)time(NULL));
  for (i=1; i<N; i=i*2)
      logn++;
  fpo=fopen("time.txt", "w");
        for(g=1; g<=G; g=g*2)
         for(d=1; d<=L; d=d*2) {
                k = ((float)(2*g*N+2*d))/(N*logn+N*2*g);  /* 最適な基準値の位置を求める */
```

```
                    tmid=0;tmax=0;
              for(i=0; i<10; i++) {
          inputArray(a,N);
  t = quickSort(a,0,N-1,0,P);
  tmid = tmid +t;
  if(tmax<t) tmax =t;

  }
  tmid = tmid /10;
  printf("基準値:%f,通信帯域幅:%3d, 通信遅延:%3d, 実行時間 :%7d,%7d ¥n", k,g,d,tmid,tmax);
    fprintf(fpo,"基準値:%f,通信帯域幅:%3d, 通信遅延:%3d, 実行時間 :%7d,%7d ¥n", k,g,d,tmid,tmax);
  }

    fclose(fpo);
}

void inputArray(a,n) int *a,n; {    /* 入力データ作成 */
  int i;

  for(i=0; i<n; i++) a[i] = rand()%M;
}

void outputArray(a,n) int *a,n; {   /* データ出力 */
  int i;

  printf("Data : ");
  for(i=0; i<n; i++) printf("%d ", a[i]);
  printf("¥n");
}

int quickSort(a,l,r,p,pp) int *a,l,r,p,pp; {
/* プロセッサpによるa[l]～a[r]間のソート */
/* 関数中でpp台のプロセッサを呼び出す */
  int i,j,t,t1,t2;
  int pivot;
  int swap();
```

```
  t=0; t1=0; t2=0;

if(logn<(r-l)/2) numofpiv=logn;
else numofpiv=(r-l)/2;
if(numofpiv==0) numofpiv=1;
for(i=0; i<=numofpiv; i++)
{
b[i] = a[(r-l)*i/(numofpiv)+l];
}
t=t+select(b,0,numofpiv-1,numofpiv/2,&pivot);
    i=l;
  j=r;
  t=t+4; /* m=, pivot=, i=, j=,*/

  do {
  t++; /* do */
  while(a[i]<pivot) {
          i++;
          t=t+2; /* while,i++ */
  }
  t++; /* while */
  while(a[j]>pivot) {
          j--;
          t=t+2; /* while,i-- */
  }
  t++; /* while */
  if(i<=j) {
          t=t+swap(a,i,j);
          i++;
          j--;
          t=t+2; /* i++,j++ */
  }
  t++; /* if */
  } while(i<=j);
  if(pp>=2) {   /* 複数のプロセッサで分担 */
```

```
  if(l<j)
     t1=quickSort(a,l,j,p,pp/2);  /* 再帰：a[l]～a[j]間のソート */
  t++; /* if */
  if(i<r)
     t2=quickSort(a,i,r,p,pp/2)+(r-i+1)*g+d; /* 再帰：a[l]～a[j]間のソート */
  t++; /* if */
  if(t1>t2) t=t+t1; else t=t+t2;
  }
  else {    /* 1台のプロセッサで全て実行 */
  if (l<j) t=quickSort(a,l,j,p,1);  /* 再帰：a[l]～a[j]間のソート */
  t++; /* if */
  if (i<r) t=t+quickSort(a,i,r,p,1); /* 再帰：a[l]～a[j]間のソート */
  t++; /* if */
           }
  t++; /* if */
  return t;
}

int select(a,l,r,k,o) int *a,l,r,k,*o; {
  int i,j,m,t,t1,t2,val;
  int pivot;
  int swap();

  t=0; t1=0; t2=0;

  m = (l+r)/2;
    pivot = a[m];
  i=l;
  j=r;
  t=t+4;

  do {
  t++;
  while(a[i]<pivot) {
           i++;
           t=t+2;
```

```
    }
    t++;
    while(a[j]>pivot) {
            j--;
            t=t+2;
    }
    t++;
    if(i<=j){
            t=t+swap(a,i,j);
            i++;
            j--;
            t=t+2;

    }
    t++;

    } while(i<=j);
    if(l<j && l<=k && k<=j) t=t+select(a,l,j,k,o);
      if(i<r && i<=k && k<=r) t=t+select(a,i,r,k,o);

    t=t+2;

    *o = a[k];
      t++;

return t;
}

int swap(a,i,j) int *a,i,j; {   /* データ入れ換え */
    int tmp;

    tmp = a[i];
    a[i] = a[j];
    a[j] = tmp;
    return 3;
}
```