卒業研究報告書

題目

BSPモデルにおける最適プロセッサ台数の推移

指導教員

石水 隆　助手

報告者

02–1–47–015

吉田鉄哉

近畿大学理工学部情報学科

平成１９年２月５日提出

概要

本研究では、BSP(Bulk-Synchrous Parallel) モデル [1] 上でソーティング行う効率の良い並列アルゴリズムの提示を行う。

BSP モデルは非同期式分散メモリ型並列計算モデルであり、並列計算において重要とされる通信コストを同期時間 L、通信命令実行時間 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。

本研究では、BSP モデル上で、バイトニックソート (Bitonic Sort)[2] を実行させたときに、通信コスト L および g が変化した際に最速となるプロセッサ数を求める。また、バイトニックソートの BSP モデル上での実行をシミュレートするプログラムを用いその計算量を実験的に評価し、理論値との比較を行う。

目 次

1 序論

1.1 並列アルゴリズム

地球規模の気象シミュレーションや天体の軌道計算など、計算量の大きな問題を短時間で解く必要のある分野は多岐に渡っている。これらの問題に対して、従来の 1 台のプロセッサから成る逐次計算機を用いた逐次処理では非常に大きな時間が掛かる。このため、これらの問題を解く手法として、複数のプロセッサを持つ並列計算機 (Parallel Computer) による並列処理 (Parallel Processing) が現在注目されている。複数のプロセッサ (Processor) が協調してデータを処理することにより、問題を短時間で解け、またより複雑な問題を解くことができるようになる。しかし、並列処理を行うためには、プロセッサ間のデータのやり取りやメモリへのアクセス、プロセッサ間の同期等、並列特有の問題を解決せねばならない。このため、従来の逐次処理で用いられてきた逐次アルゴリズムをそのまま並列処理に用いることはできず、並列処理専用のアルゴリズム、すなわち並列アルゴリズム (Parallel Algorithm) が必要となる。そのため、現在様々な分野で、高速に処理を行う並列アルゴリズムが求められている。

1.2 並列計算機

並列計算機は複数のプロセッサを持ち並列処理を行うことができる計算機である。並列計算機は、全てのプロセッサが共通したメモリに対して読み書きを行い、プロセッサ間の通信はメモリを通して行う共有メモリ型並列計算機と、それぞれのプロセッサが局所メモリを持ち、プロセッサ間の通信はネットワークを通じて行う分散メモリ型並列計算機に大別される。プロセッサ数の増加に従い、1 つの共有メモリに全てのプロセッサを繋ぐことは困難となる。このため、現在、プロセッサ数の多い並列計算機では分散メモリ型が主流となっている。また、複数の計算機をネットワークで繋ぎ、それ全体を仮想的な計算機として扱うクラスタ (Cluster) 処理やグリッド (Grid) 処理も幅広く行われている。

1.3 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。代表的な並列計算モデルとして、PRAM(Parallel Random Access Machine), Mesh, Hyper-cube, BSP (Bulk-Syncronous Parallel) モデル, CGM (Coarse Grain Multi-Computer) などがある。

PRAM は共有メモリ型並列計算モデルであり、全ての演算が 1 単位時間で行われる、1 命令毎に同期が取られる、通信のコストが一切発生しない、等の仮定が設けられた理想的なモデルである。このため PRAM 上でのアルゴリズムの設計・解析は比較的容易に行うことができる。しかし、PRAM 自体の実現は困難であり、PRAM 上で設計したアルゴリズムは現実の並列計算機では必ずしも効率良く実行できるとは限らない。このため、現在主流となってきた分散メモリ型並列計算機に対応するモデルとして注目されているのは BSP モデルである。BSP モデルは分散メモリ型並列計算モデルであり、通信のオーバヘッドや同期のオーバヘッドを考慮することができるモデルである。そこで本研究ではモデルとして BSP モデルを採用し、この上で高速に実行できる並列アルゴリズムの設計を行う。

1.4 ソーティング

ソーティングは基本的な問題であり、様々な分野で広く用いられる。このため、並列計算機上で高速にソーティングを解くことができる並列アルゴリズムを開発することは重要な課題である。サイズ n のデータに対し、逐次アルゴリズムでは、クィックソート (Quick Sort) やマージソート (Marge Sort) を用いて $O(\log n)$ 時間でソーティングを行うことができる。並列アルゴリズムでは、バイトニックソート (Bitonic Sort) により p プロセッサ EREW-PRAM を用いて $O(\frac{n \log n \log p}{p} + \log^2 p)$ $(0 \leq p < n)$ 時間でソーティングを行うことができる。また、バイトニックソートを BSP モデル上で実行させることにより $O(\frac{n \log n \log p}{p} + (\frac{gn}{p} + L) \log^2 p)$ 時間 $(0 \leq p < n)$ でソーティングを行うことができる。ただし、L は同期時間、g は通信命令実行時間である。

1.5 本報告書の構成

本報告書の構成を以下に述べる。2 章では本研究で使用する BSP モデルとバイトニックソートについて述べ、最後にシミュレートプログラムを用いて最適プロセッサを求める手順について述べる。**??**章では最適なプロセッサ数を求めて得られた結果とシミュレータと理論値を比較した結果と考察について述べる。**??**章では得られた結果から導き出した結論、及び今後の課題について述べる。

2 準備

2.1 BSP モデル

BSP(Bulk-Synchronous Parallel) モデル [(1]はブロック体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 (本論文中ではプロセッサ数を p とし，各プロセッサを P_i $(1 \leq i \leq p)$ で表す．)
- 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

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。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[1]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。

BSP モデルは以下の 2 つのパラメタにより，具体的なネットワーク構造やメッセージ配送の仕組みを抽象化している。

- L:バリア同期周期
- g $(\leq L)$:1 個の送信命令または受信命令の実行に必要な時間

BSP モデル上の並列アルゴリズムの基本的命令の実行時間について、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 単位時間で行なわれる。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。
- 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 1 に BSP モデル上でのアルゴリズムの実行の概念図を示す。

以降では簡単のために、各スーパーステップは内部計算命令のみ、あるいは送信命令および受信命令のみからなるとし，内部計算命令のみからなるスーパーステップの実行時間を内部計算時間、送信命令および受信命令のみからなるスーパーステップの実行時間を通信時間と呼ぶ。以下では、内部計算に掛かる時間を T_I、通信命令に掛かる時間を T_C、同期に掛かる時間を T_S と記述し、アルゴリズムの実行全体に掛かる時間を $T = T_I + T_C + T_S$ と記述する。

[1]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サッサも実行を継続できず、封鎖される。

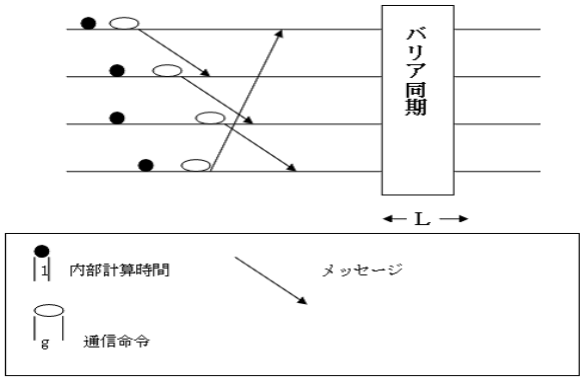

図 1: BSP アルゴリズム上での並列計算の動き

2.2 バイトニックソート

以下に BSP モデル上でのバイトニックソートアルゴリズムを示す。

入力 n 個のデータ A。各プロセッサ P_i が $\frac{n}{p}$ 個のデータから成る A の部分データ A_i を保持する。

出力 ソート済みの n 個のデータ B。各プロセッサ P_i が $\frac{n}{p}$ 個のデータから成る B の部分データ B_i を保持する。

(1) 各プロセッサ P_i $(0 \leq i < n)$ は逐次アルゴリズムを用いて保持するデータをソートする。

(2) 以下の操作を $\log p$ 回繰り返す。なお、k回目の繰り返しにおいて、プロセッサは2^{k-1} 個ずつ $\frac{p}{2^{k-1}}$ 個のグループ $G_{k,0}, G_{k,1}, ...G_{k,\frac{p}{2^{k-1}}-1}$ に分けられているとし、繰り返し開始時点において各グループ $G_{k.m}$ $(0 \leq m < \frac{p}{2^{k-1}})$ 内のデータはソート済みであるとする。

(2.1) プロセッサグループ $G_{k,m}$ $(0 \leq m < \frac{p}{2^{k-1}})$ を 2 グループずつ組 $(G_{k,2n}, G_{k,2n+1})$ $(0 \leq n < \frac{p}{2^k})$ の組とする。

(2.2) 各グループ $G_{k,2n}, G_{k,2n+1}$ においてグループ内で保持するデータのうち小さいものを $G_{k,2n}$ に、大きいものを $G_{k,2n+1}$ に送信する。

(2.3) バイトニックマージを用いて各グループ内のデータをソートする。

2.3 シミュレートプログラムによる検証法

本節では、通信コストが変化したときのバイトニックソートの最適なプロセッサ台数をシミュレートプログラムを用いて、求める手順について述べる。以下ではシミュレートプログラムにより計算された内部計算時間を S_I、通信時間を S_C、同期時間を S_S、アルゴリズム全体の実行時間を S と表記する。付録 1 に示すシミュレートプログラムより、バイトニックソートの進行時間が以下の式で表されることが示される。ただし、d=n/p とする。

$$
\begin{aligned}
S_I &= d \log d(\frac{log^2 p}{2} + 1) \\
S_C &= gd\frac{\log^2 p}{2} \\
S_S &= L\frac{\log^2 p}{2} \\
S &= S_I + S_C + S_S
\end{aligned}
$$

同期時間 S_S が $L\frac{\log^2 p}{2}$ であるので、シミュレートプログラムでは $\frac{\log^2 p}{2}$ 回、同期が行われている。また、プロセッサ台数 p が 1 のとき S=n$\log n$ となり、逐次ソートの計算量に一致する。

通信コスト L,g に対して最速になるプロセッサ台数 p を求めるため、プロセッサ台数 p のときの実行時間 S(p) と 2p のときの実行時間 S(2p) を比較し $S(p) < S(2p)$ となる p を求める。通信コスト L,g に対して最速になるプロセッサ台数

p を求めるため、プロセッサ台数 p のときの実行時間 S(p/2) と p のときの実行時間 S(p) を比較し $S(p/2) < S(p)$ となる p を求める。$S(\frac{p}{2})$,S(p) の差を $\mathrm{F(p,n)} = \mathrm{S}(\frac{\mathrm{p}}{2}) - \mathrm{S(p)}$ とする
このとき、$S(\frac{p}{4}) > S(\frac{p}{2}) < S(p)$ を満たす L は次の式で与えられる。

$$\mathrm{L} = (\frac{2\mathrm{F(p)}}{1 - 2\log \mathrm{p}}) + 2 \tag{1}$$

また、p=1 のときよりも実行時間が多くなる g,L の値は、次の式で与えられる。

$$\mathrm{g} = \mathrm{S}(1,\mathrm{n}) - \frac{\mathrm{S(p,n)}}{\mathrm{n}\log^2 \mathrm{p}}p + 2 = \mathrm{n}\log \mathrm{n} - \frac{\mathrm{S(p,n)}}{\mathrm{n}\log^2 \mathrm{p}}p + 2 \tag{2}$$

$$\mathrm{L} = \mathrm{S}(1,\mathrm{n}) - \frac{\mathrm{LS(p,n)}}{\log^2 \mathrm{p}} = \mathrm{n}\log \mathrm{n} - \frac{\mathrm{LS(p,n)}}{\log^2 \mathrm{p}} \tag{3}$$

(1),(2),(3) の式を使い最速の計算量を持つプロセッサを求めることが出来る。以下に最適なプロセッサの求め方を示す。

1. まず扱うデータ数 n を定め、その n での最適プロセッサを求める。
2. (2) の式から任意の p と d=2 のときの g の値がその p,n の上限の g となる。
3. (3) の式から同 d 上で最も高い L を持つプロセッサ p 以上の数のプロセッサが L がこの式で分かった最も高い L 以下の範囲で p1 以外で最適となるプロセッサである。そしてその時の L の値がその p が最適なプロセッサである上限 L となる。
4. (3) で求めた p まで (1) の式を使いそれぞれのプロセッサが最適である際の上限 L を求める。

3 結果と考察

本研究で示された結果について述べる。簡単のために以下では n=512 の場合について説明する。

3.1 シミュレータによる値

まず (2) の式から d=2,p256 で計算した値を求める。これがプロセッサが 1 台より上の台数のプロセッサが最適なプロセッサ台数となりうる g の上限。プロセッサ 1 台より計算時間が長くなる g を表 1 に示す

	d2	d4	d8	d16	d32	d64	d128
p2	0	0	0	0	0	0	0
p4	0	0	0	0	0	0	0
p8	2	2	2	2	2	2	2
p16	5	5	5	5	5	5	5
p32	9	9	10	11	11	12	12
p64	16	17	19	20	21	23	24
p128	28	31	34	37	39	42	44
p256	51	56	61	66	71	76	81

表 1: プロセッサ 1 台より計算時間が長くなる g

表 1 からこの場合 g の上限は 51 だと分かる（それ以上の時は、プロセッサ 1 台が最適な台数となる）。

次に g=1 と定め、(3) の式から n=512 として p2 から p256 までの L の値を計算する。この値を表 2 に示す。この n512 の列の中から最も大きい L の値を持つプロセッサと L の値を調べる（この表では p32,L148)。そのプロセッサから下のプロセッサがその g の値で最適なプロセッサになり得るプロセッサである。これはこのプロセッサより上の値はプロセッサ

	n2	n4	n8	n16	n32	n64	n128	n256	n512
p1	0	0	0	0	0	0	0	0	0
p2		0	0	0	0	0	0	0	0
p4			2	0	0	0	0	0	0
p8				3	5	9	13	18	22
p16					8	15	31	62	126
p32						16	33	70	148
p64							30	64	138
p128								55	120
p256									101

表 2: プロセッサ 1 台より計算時間が長くなる L

	n8	n16	n32	n64	n128	n256	n512
p4	4	7	14	27	54	107	214
p8		8	18	39	85	185	401
p16			12	28	63	141	314
p32				16	38	87	198
p64					20	48	111
p128						24	58
p256							28

表 3: 最適なプロセッサ台数となる L の値

は元の値が高く最適プロセッサにならないのが一つ。あと一つはこれより下のプロセッサは L が上昇することによりプロセッサが多いほうが上昇率が高い。そのため L が上昇するごとに最適プロセッサは多い方から少ない方へ推移する。

表 3 より g=1 の時、L＜２８なら最適プロセッサ数ｐ２５６、２８＜＝Ｌ＜５８ならｐ１２８、５８＜＝Ｌ＜１１１ならｐ６４、１１１＜＝Ｌ＜１４８ならｐ３２と求まる。この三つの式を使えば任意の n,g での L が推移することによる、最適プロセッサ数を求めることが可能である L の増加による計算量の推移を図 2 に示す。

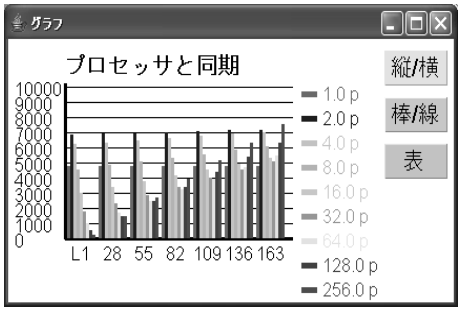

図 2: L が 27 毎に増加する際のプロセッサの計算量の推移

図 2 で示される L の値での最適プロセッサは、前述の L の範囲での最適プロセッサと一致する。

以下は式から得られた値の推移における特徴である。
d と g が増加することによる (1) 式の L の値の増加量はおよそ　$(2+g+\log d)d/2$ である。従って表 4 に示す最適なプロセッサ台数となる L の値が得られる。

表 4 では g=1 の表で d2 の場合とは (p4,n8),(p8,n16),(p16,n32) の部分 (この表で (p4,n8) から (p8,n16)、(p8,n16) か

	n8	n16	n32	n64	n128	n256	n512
p4	4	7	14	27	54	107	214
p8		8	18	39	85	185	401
p16			12	28	63	141	314
p32				16	38	87	198
p64					20	48	111
p128						24	58
p256							28

	n8	n16	n32	n64	n128	n256	n512
p4	3	6	11	22	43	86	171
p8		9	19	41	89	193	417
p16			14	31	69	154	340
p32				19	43	98	219
p64					24	55	126
p128						29	68
p256							34

	n8	n16	n32	n64	n128	n256	n512
p4	3	5	9	17	33	65	129
p8		9	20	43	93	201	433
p16			15	34	76	167	365
p32				22	49	109	241
p64					28	63	141
p128						34	77
p256							40

表 4: 最適なプロセッサ台数となる L の値、g=1(上)、g=2(中)、g=3(下)

ら (p16,n32) の部分) で増加する L の値が (2+1+log 2)2/2 = 4 の値に相当する。n が多いほどこの式の値に近づく。n が少ないほど誤差が大きい (式の値よりも高い)。g が大きくなれば誤差が少なくなるまでの n の量も大きくなる。

また、プロセッサ 1 台のときの実行時間と比較することにより、表 5 に示すプロセッサ 1 台よりも実行時間が長くなる L の値が得られる。

	n2	n4	n8	n16	n32	n64	n128	n256	n512
p1	0	0	0	0	0	0	0	0	0
p2		0	0	0	0	0	0	0	0
p4			2	0	0	0	0	0	0
p8				3	5	9	13	18	22
p16					8	15	31	62	126
p32						16	33	70	148
p64							30	64	138
p128								55	120
p256									101

	n2	n4	n8	n16	n32	n64	n128	n256	n512
p1	0	0	0	0	0	0	0	0	0
p2		0	0	0	0	0	0	0	0
p4			0	0	0	0	0	0	0
p8				2	2	0	0	0	0
p16					6	11	23	46	94
p32						14	29	62	132
p64							28	60	130
p128								53	116
p256									99

	n2	n4	n8	n16	n32	n64	n128	n256	n512
p1	0	0	0	0	0	0	0	0	0
p2		0	0	0	0	0	0	0	0
p4			0	0	0	0	0	0	0
p8				0	0	0	0	0	0
p16					4	7	15	30	62
p32						12	25	54	116
p64							26	56	122
p128								51	112
p256									97

表 5: 各プロセッサがプロセッサ 1 台の時の計算量を超える L の値、g=1(上)、g=2(中)、g=3(下)

3.2 シミュレータによる値と理論値との総合計算量の比較

本節では、理論値とシミュレータによる値との比較を行う。理論値とシミュレータの計算量を比較するにあたって、まず 2 つのバイトニックソートの方法を図に示す。

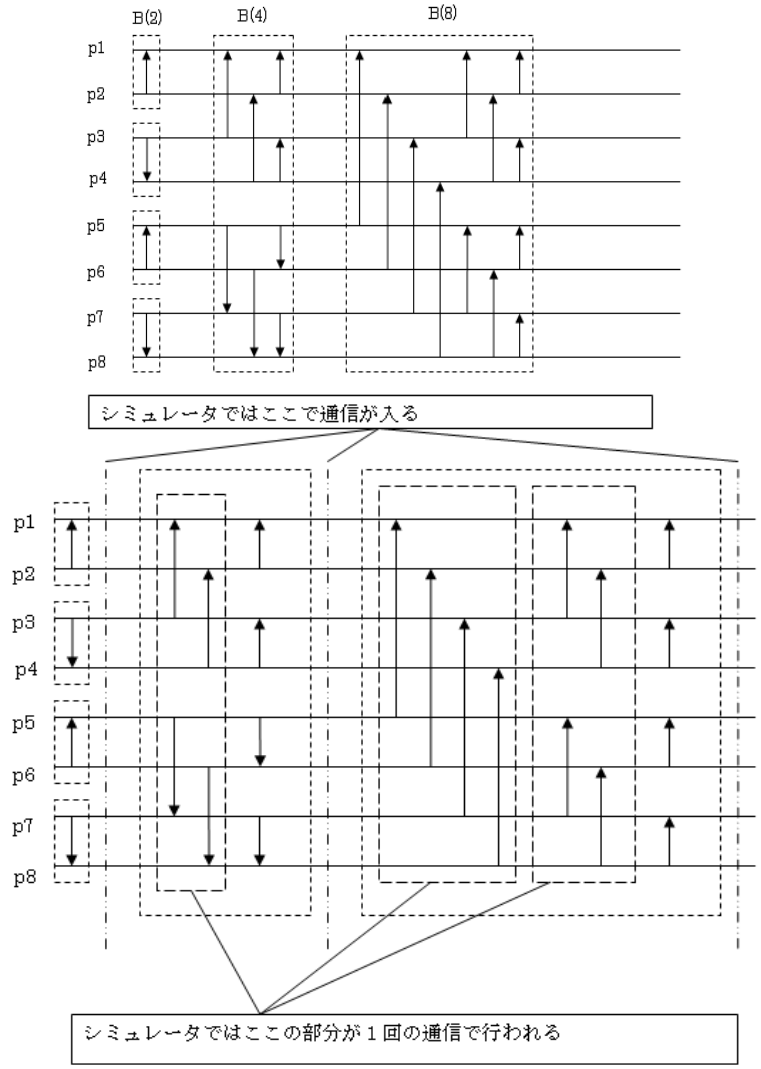

図 3: 理論値のバイトニックソート (上)、シミュレータのバイトニックソート (下)

理論値の計算量は以下の式で得られる。

$$
\begin{aligned}
\mathrm{T} &= \mathrm{T_I} + \mathrm{T_C} + \mathrm{T_S} \\
\mathrm{T_I} &= \mathrm{n}(\log \mathrm{n} + \log^2 \mathrm{p})/\mathrm{p} \\
\mathrm{T_C} &= (\mathrm{gd}) \log^2 \mathrm{p} \\
\mathrm{T_S} &= \mathrm{L} \log^2 \mathrm{p}
\end{aligned}
$$

シミュレータの計算量は以下の式で得られる。

$$
\begin{aligned}
\mathrm{S} &= \mathrm{S_I} + \mathrm{S_C} + \mathrm{S_S} \\
\mathrm{S_I} &= d \log d(\tfrac{log^2 p}{2} + 1) \\
\mathrm{S_C} &= gd\tfrac{\log^2 p}{2} \\
\mathrm{S_S} &= L\tfrac{\log^2 p}{2}
\end{aligned}
$$

表 6,7,8 にシミュレータによる値と理論地を示す。
g=1,L=1 の場合 d=2 の場合を除いて全般的に理論値の計算結果のほうが速い。
g を上げていくと以下の条件を除きシミュレータのほうが速くなっていく。

	n2	n4	n8	n16	n32	n64	n128	n256	n512
p1	2	8	24	64	160	384	896	2048	4608
p2		12	34	90	226	546	1282	2946	6658
p4			27	73	189	469	1125	2629	6021
p8				47	125	321	793	1897	4425
p16					72	190	486	1196	2862
p32						102	268	684	1684
p64							137	359	915
p128								177	463
p256									222

	n2	n4	n8	n16	n32	n64	n128	n256	n512
p1	2	8	24	64	160	384	896	2048	4608
p2		9	21	49	113	257	577	1281	2817
p4			26	52	108	228	484	1028	2180
p8				53	101	201	409	841	1737
p16					90	168	328	656	1328
p32						137	253	489	969
p64							194	356	684
p128								261	477
p256									338

表 6: g=1,L=1 のシミュレータの値 (上)、g=1,L=1 の理論値の値 (下)

- p1 から p4 までの範囲。
- p8 で n32 以降。

	n2	n4	n8	n16	n32	n64	n128	n256	n512
p1	2	8	24	64	160	384	896	2048	4608
p2		208	426	874	1794	3682	7554	15490	31746
p4			517	1053	2149	4389	8695	18309	37381
p8				929	1889	3849	7849	16009	32649
p16					1444	2934	5974	12174	24814
p32						2062	4188	8524	17364
p64							2783	5651	11499
p128								3607	7323
p256									4534

	n2	n4	n8	n16	n32	n64	n128	n256	n512
p1	2	8	24	64	160	384	896	2048	4608
p2		107	217	441	897	1825	3713	7553	15361
p4			418	836	1676	3364	6756	13572	27268
p8				935	1865	3729	7465	14953	29961
p16					1658	3304	6600	13200	26416
p32						2587	5153	10289	20569
p64							3722	7412	14796
p128								5063	10081
p256									6610

表 7: g=50,L=1 のシミュレータの値 (上)、g=50,L=1 の理論値の値 (下)

	n2	n4	n8	n16	n32	n64	n128	n256	n512
p1	2	8	24	64	160	384	896	2048	4608
p2		2010	2032	2088	2224	2544	3280	4944	8656
p4			5022	5068	5184	5464	6120	7624	11016
p8				9038	9116	9312	9784	10888	13416
p16					14058	14176	14472	15184	16848
p32						20082	20248	20664	21664
p64							27110	27332	27888
p128								35142	35428
p256									44176

	n2	n4	n8	n16	n32	n64	n128	n256	n512
p1	2	8	24	64	160	384	896	2048	4608
p2		1008	1020	1048	1112	1256	1576	2280	3816
p4			4022	4048	4104	4224	4480	5024	6176
p8				9044	9092	9192	9400	9832	10728
p16					16074	16152	16312	16640	17312
p32						25112	25228	25464	25944
p64							36158	36320	36648
p128								49212	49428
p256									64274

表 8: g=1,L=1000 のシミュレータの値 (上)、g=1,L=1000 の理論値の値 (下)

3.3 考察

シミュレータと理論値の比較をした結果、g と L の値が高ければ高いほど (1 < p <= 4) の場合と (p8,n16) の場合には、シミュレータの計算方法がよりも速くなっていってしまう傾向があることが分かった。

4 結論および今後の課題

本研究では、BSP モデル上でバイトニックソートを実行させたとき最適となるプロセッサの台数を、理論値およびシミュレートプログラムによる値の両面から求めた。本研究で求めたプロセッサ台数を用いることにより効率よくソーティングを行うことができる。

しかし、理論値とシミュレータにより求まる値との間にはずれがあるため、このずれを解消できるシミュレータを作成することが今後の課題である。

5 参考文献

1) L.G.Valiant, "A Bridging Model for Parallel Computation, ", Communications of the ACM, Vol.33, No.8, pp.103–111, 1990.

2) J.JáJá, "An Introduction to Parallel Algorithms," Addison-Wesley Publishing Company, 1999.

3) 静岡理工科大学　　菅沼研究室
http://www.sist.ac.jp/ suganuma/main.htm

A 付録

BspMergeSort.java

```
import ioTools.*;  //ファイル入出力用
import java.io.*;  //ファイル入出力用

public class BspMergeSort {
static final int gap = 1;                    // 通信コスト
static final int latency = 1;                // 通信遅延
static final int numOfProcessors = 8;        // プロセッサ数
static final int numOfData = 16;             // データ数

static BspProcessor[] processor;             // プロセッサ
static BspNetwork network;                   // ネットワーク
static InputData input;                      // 入力データ

static boolean debugSW = false;              // デバグ用スイッチ
static boolean traceSW = false;              // トレス用スイッチ

static PrintWriter output;                   // 出力用ファイル
static PrintWriter log;                      // ログ出力用ファイル

public static void main (String[] args) {
// ネットワークを作る
network = new BspNetwork (numOfProcessors);

// プロセッサを作る
processor = new BspProcessor[numOfProcessors];
for (int p=0; p<numOfProcessors; p++) {        // プロセッサ i を作る
processor[p] = new BspProcessor (p, numOfProcessors);
processor[p].setNetwork (network);
}

// ランダムデータ作成
input = new InputData (numOfData);
input.makeRandomData(); // ソート済データが必要なら input.makeSortedData() を用いる
if (traceSW) input.dump();
input.dumpToFile();

// 各プロセッサが保持する初期値設定
for (int p=0; p<numOfProcessors; p++) {
int low = p * numOfData / numOfProcessors;
int high = (p+1) * numOfData / numOfProcessors - 1;
processor[p].setData(input.get(low, high));
}
```

```
// 時計初期化
for (int i=0; i<numOfProcessors; i++)
processor[i].setTime(0,0,0);

// 出力用ファイル設定
output = FileIo.fWrite("OutputData.txt", false);
log = FileIo.fWrite("BspMergeSort.log", false);
network.setLogFile(log);
for (int p=0; p<numOfProcessors; p++) {
processor[p].setOutputFile(output);
processor[p].setLogFile(log);
}

// 初期データ表示
System.out.println("Input Data");
for (int p=0; p<numOfProcessors; p++) {
processor[p].showData();
processor[p].logData();
}

// ソーティング
sequentialMergeSort();
bitonicMergeSort();

// ソート後データ表示
System.out.println("Output Data");
for (int p=0; p<numOfProcessors; p++) {
processor[p].showData();
processor[p].printData();
processor[p].logData();
}

// 実行時間表示
showTime();
printTime();
logTime();

// 出力用ファイルを閉じる
output.close();
log.close();
}

// 並列マージソート
public static void bitonicMergeSort () {
for (int i=2; i<=numOfProcessors; i*=2) { // log numOfProcessor 回繰り返す

// プロセッサ low〜プロセッサ high-1 がそれぞれのデータを 2 つに分割したとき、
```

```
// それぞれの部分データの送り先のプロセッサ (lowProcessor, highProcessor) を選ぶ
// lowProcessor に小さい部分の送り先、highProcessor に大きい部分の送り先が入る
// ただし、前半のプロセッサはプロセッサ順の通りに送り先を決定し、
// 後半のプロセッサはプロセッサ順の逆順に送り先を決定する
if (debugSW) System.out.println("selectSendingProcessorsBitonic");
log.println("selectSendingProcessorsBitonic");
for (int j=0; j<numOfProcessors; j+=i)
for (int p=j; p<j+i; p++)
processor[p].selectSendingProcessorsBitonic (j, j+i);
if (traceSW)
for (int p=0; p<numOfProcessors; p++)
processor[p].showReceivers();
for (int p=0; p<numOfProcessors; p++)
processor[p].logReceivers();

// 前半のデータを lowProcessor に、後半のデータを highProcecceor に送る
// ただし、前半のプロセッサは昇順に、後半のプロセッサは降順に送る
if (debugSW) System.out.println("sendDataBitonic");
log.println("sendDataBitonic");
for (int j=0; j<numOfProcessors; j+=i)
for (int p=j; p<j+i; p++)
processor[p].sendDataBitonic(j, j+i);

synchronous(); // 同期

// データを受信する
// (受信したデータは前半が昇順、後半が降順になっている)
if (debugSW) System.out.println("receiveData");
log.println("receiveData");
for (int p=0; p<numOfProcessors; p++)
processor[p].receiveData();

// データをマージする
if (debugSW) System.out.println("mergeData");
log.println("mergeData");
for (int p=0; p<numOfProcessors; p++)
processor[p].mergeData();
if (traceSW)
for (int p=0; p<numOfProcessors; p++)
processor[p].showData();
for (int p=0; p<numOfProcessors; p++)
processor[p].logData();

// プロセッサ low〜プロセッサ high-1 がそれぞれのデータを 2 つに分割したとき、
// それぞれの部分データの送り先のプロセッサ (lowProcessor, highProcessor) を選ぶ
// lowProcessor に小さい部分の送り先、highProcessor に大きい部分の送り先が入る
// ただし、全てのプロセッサでプロセッサ順の通りに送り先を決定する
```

```
if (debugSW) System.out.println("selectSendingProcessors");
log.println("selectSendingProcessors");
for (int j=0; j<numOfProcessors; j+=i)
for (int p=j; p<j+i; p++)
processor[p].selectSendingProcessors (j, j+i);
if (traceSW)
for (int p=0; p<numOfProcessors; p++)
processor[p].showReceivers();
for (int p=0; p<numOfProcessors; p++)
processor[p].logReceivers();

for (int j=i; j>1; j/=2) {    // log i 回繰り返す
// i<numOfProcessor なので、全体として (log numOfProcessor)^2 回繰り返す)

// 前半のデータを lowProcessor に、後半のデータを highProcecceor に送る
// ただし、前半のプロセッサは昇順に、後半のプロセッサは降順に送る
if (debugSW) System.out.println("sendDataBitonic");
log.println("sendDataBitonic");
for (int k=0; k<numOfProcessors; k+=i)
for (int p=k; p<k+i; p++)
processor[p].sendDataBitonic(k, k+i);

synchronous(); // 同期

// データを受信する
// (受信したデータは前半が昇順、後半が降順になっている)
if (debugSW) System.out.println("receiveData");
log.println("receiveData");
for (int p=0; p<numOfProcessors; p++)
processor[p].receiveData();

// データをマージする
if (debugSW) System.out.println("mergeData");
log.println("mergeData");
for (int p=0; p<numOfProcessors; p++)
processor[p].mergeData();
if (traceSW)
for (int p=0; p<numOfProcessors; p++)
processor[p].showData();
for (int p=0; p<numOfProcessors; p++)
processor[p].logData();
}
}
}

// 逐次マージソート
public static void sequentialMergeSort () {
```

```
if (debugSW) System.out.println("sequentialMergeSort");
log.println("sequentialMergeSort");
for (int p=0; p<numOfProcessors; p++)
processor[p].sequentialMergeSort();
if (traceSW)
for (int p=0; p<numOfProcessors; p++)
processor[p].showData();
for (int p=0; p<numOfProcessors; p++)
processor[p].logData();
}

// 全てのプロセッサ間の同期
public static void synchronous () {
synchronous (0, numOfProcessors-1);
}

// プロセッサ p〜プロセッサ q 間の同期
public static void synchronous (int p, int q) {
if (traceSW)
for (int i=p; i<=q; i++)
network.showSendQueue(i);
for (int i=p; i<=q; i++)
network.logSendQueue(i);

// ネットワークの送信キュー内のデータを受信キューに移す
network.synchronous(p, q);

int timeI = 0;
int timeG = 0;
int timeL = 0;
for (int i=p; i<=q; i++) { // プロセッサ p〜プロセッサ q の時間を最大値に揃える
if (processor[i].timeI > timeI) timeI = processor[i].timeI;
if (processor[i].timeG > timeG) timeG = processor[i].timeG;
if (processor[i].timeL > timeL) timeL = processor[i].timeL;
}

timeL++;

for (int i=p; i<=q; i++)
processor[i].setTime(timeI, timeG, timeL);
}

// 実行時間表示
public static void showTime() {
System.out.println("time : " + processor[0].timeI + " + g*" + processor[0].timeG + " + L*" + processor[0].
}
```

```
// 実行時間出力
public static void printTime() {
output.println("time : " + processor[0].timeI + " + g*" + processor[0].timeG + " + L*" + processor[0].time
}

// 実行時間出力(ログ出力用)
public static void logTime() {
log.println("time : " + processor[0].timeI + " + g*" + processor[0].timeG + " + L*" + processor[0].timeL);
}
}
```

BspNetwork.java

```
import java.util.ArrayList;  // ArrayList 処理用
import ioTools.*;  //ファイル入出力用
import java.io.*;  //ファイル入出力用

public class BspNetwork {
ArrayList[] sendQueue;          // プロセッサ i へ送信中のデータ
ArrayList[] receiveQueue;       // プロセッサ i が受信中のデータ

PrintWriter log;                // ログ出力用ファイル

// プロセッサ p 台を結ぶネットワークを作る
public BspNetwork(int p) {
sendQueue = new ArrayList[p];
receiveQueue = new ArrayList[p];
for (int i=0; i<p; i++) {
sendQueue[i] = new ArrayList();
receiveQueue[i] = new ArrayList();
}
}

// int 型データ n をプロセッサ p への送信キューに置く
public void put (int p, int n) {
sendQueue[p].add(new Integer(n));
}

// int 型配列データ a をプロセッサ p への送信キューに置く
public void putArray (int p, int[] a) {
for (int i=0; i<a.length; i++)
sendQueue[p].add(new Integer(a[i]));
}

// 2 個組 int 型データ (m,n) をプロセッサ p への送信キューに置く
public void putPair(int p, int m, int n) {
sendQueue[p].add(new Integer(m));
sendQueue[p].add(new Integer(n));
}

// int 型配列データ a の a[l]〜a[h] をプロセッサ p への送信キューに置く
// l,h が 0 未満または a.length 以上のときはエラー
public void putPartOfArray (int p, int[]a, int l, int h) {
if (l<0 || l>=a.length || h<0 || h>=a.length)
executeError("Illegal index of array at Queue "+p);
for (int i=l; i<=h; i++) {
sendQueue[p].add(new Integer(a[i]));
}
```

```
}

// int 型配列データ a の a[l]〜a[h] をプロセッサ p への送信キューに逆順 (a[h]〜a[l]) に置く
// l,h が 0 未満または a.length 以上のときはエラー
public void putPartOfArrayReverce (int p, int[]a, int l, int h) {
if (l<0 || l>=a.length || h<0 || h>=a.length)
executeError("Illegal index of array at Queue "+p);
for (int i=h; i>=l; i--) {
sendQueue[p].add(new Integer(a[i]));
}
}

// int 型行列データ m をプロセッサ p への送信キューに置く
public void putMatrix (int p, int[][]m) {
for (int i=0; i<m.length; i++)
for (int j=0; j<m[i].length; j++)
  sendQueue[p].add(new Integer(m[i][j]));
}

// プロセッサ p への受信キューから int 型データを取り出す
// 受信キューにデータが入っていない場合、取り出したデータが Integer 型でない場合はエラー
public int get (int p) {
if (receiveQueue[p].isEmpty()) executeError("Queue "+p+" Underflow");
if ((receiveQueue[p].get(0)).getClass() !=  Integer.class) executeError("Queue "+p+" Type Mismatched");
int n = ((Integer) receiveQueue[p].get(0)).intValue();
receiveQueue[p].remove(0);
return n;
}

// プロセッサ p への受信キューから int 型配列データを取り出す
// 受信キューにデータが入っていない場合、取り出したデータが Integer 型でない場合はエラー
public int[] getArray (int p) {
if (receiveQueue[p].isEmpty()) executeError("Queue "+p+" Underflow");
int[] a = new int[receiveQueue[p].size()];
for (int i=0; i<a.length; i++) {
if ((receiveQueue[p].get(0)).getClass() !=  Integer.class) executeError("Queue "+p+" Type Mismatched");
a[i] = ((Integer) receiveQueue[p].get(0)).intValue();
receiveQueue[p].remove(0);
}
return a;
}

// プロセッサ p への受信キューから 2 個組 int 型データを取り出す
// 受信キューに 2 個以上データが入っていない場合、取り出したデータが Integer 型でない場合はエラー
public int[] getPair (int p) {
if (receiveQueue[p].size() < 2) executeError("Queue "+p+" Underflow");
if ((receiveQueue[p].get(0)).getClass() !=  Integer.class) executeError("Queue "+p+" Type Mismatched");
```

```
int m = ((Integer) receiveQueue[p].get(0)).intValue();
receiveQueue[p].remove(0);
if ((receiveQueue[p].get(0)).getClass() !=  Integer.class) executeError("Queue "+p+" Type Mismatched");
int n = ((Integer) receiveQueue[p].get(0)).intValue();
receiveQueue[p].remove(0);
int[] intPair = {m,n};
return intPair;
}

// プロセッサ p への受信キューから長さ s の int 型配列データを取り出す
// 受信キューに s 個以上データが入っていない場合、取り出したデータが Integer 型でない場合はエラー
public int[] getArray (int p, int s) {
if (receiveQueue[p].size() < s) executeError("Queue "+p+" Underflow");
int[] a = new int[s];
for (int i=0; i<s; i++) {
if ((receiveQueue[p].get(0)).getClass() !=  Integer.class) executeError("Queue "+p+" Type Mismatched");
a[i] = ((Integer) receiveQueue[p].get(0)).intValue();
receiveQueue[p].remove(0);
}
return a;
}

// プロセッサ p への受信キューからサイズ s × s の int 型行列データを取り出す
// 受信キューに s*s 個以上データが入っていない場合、取り出したデータが Integer 型でない場合はエラー
public int[][] getMatrix (int p, int s) {
if (receiveQueue[p].size() < s*s) executeError("Queue "+p+" Underflow");
int[][] m = new int[s][s];
for (int i=0; i<s; i++)
for (int j=0; j<s; j++) {
if ((receiveQueue[p].get(0)).getClass() !=  Integer.class) executeError("Queue "+p+" Type Mismatched");
m[i][j] = ((Integer) receiveQueue[p].get(0)).intValue();
receiveQueue[p].remove(0);
}
return m;
}

// プロセッサ p への受信キューからサイズ s × t の int 型行列データを取り出す
// 受信キューに s*t 個以上データが入っていない場合、取り出したデータが Integer 型でない場合はエラー
public int[][] getMatrix (int p, int s, int t) {
if (receiveQueue[p].size() < s*s) executeError("Queue "+p+" Underflow");
int[][] m = new int[s][t];
for (int i=0; i<s; i++)
for (int j=0; j<t; j++) {
if ((receiveQueue[p].get(0)).getClass() !=  Integer.class) executeError("Queue "+p+" Type Mismatched");
m[i][j] = ((Integer) receiveQueue[p].get(0)).intValue();
receiveQueue[p].remove(0);
}
```

```
return m;
}

// プロセッサ p〜プロセッサ q の送信キューのデータを受信キューに移す
// (この操作を行わないと受信キューからデータを取り出せない)
public void synchronous(int p, int q) {
for (int i=p; i<=q; i++) {
while (!(sendQueue[i].isEmpty())) { // 送信キューが空になるまで
receiveQueue[i].add(sendQueue[i].get(0));
sendQueue[i].remove(0);
}
}
}

// プロセッサ p への受信キューをクリアする
public void clear (int p) {
receiveQueue[p].clear();
}

// プロセッサ p への受信キューが空であるか?
public boolean isEmpty (int p) {
return receiveQueue[p].isEmpty();
}

// プロセッサ p への受信キューに置かれたデータの数を返す
public int size (int p) {
return receiveQueue[p].size();
}

// ログ出力用ファイルをセットする
public void setLogFile(PrintWriter l) {
log = l;
}

// プロセッサ p への送信キューの内容を表示する (デバグ用)
public void showSendQueue (int p) {
if (p < 10)
System.out.print ("Pr. "+p+":");
else System.out.print ("Pr."+p+":");
System.out.println (formatQueue (sendQueue[p]));
}

// プロセッサ p への受信キューを表示する (デバグ用)
public void showReceiveQueue (int p) {
if (p < 10)
System.out.print ("Pr. "+p+":");
else System.out.print ("Pr."+p+":");
```

```
System.out.println (formatQueue (receiveQueue[p]));
}

// プロセッサ p への送信キューをログファイルに出力する
public void logSendQueue (int p) {
if (p < 10)
log.print ("Pr. "+p+":");
else log.print ("Pr."+p+":");
log.println (formatQueue (sendQueue[p]));
}

// プロセッサ p の受信キューをログファイルに出力する
public void logReceiveQueue (int p) {
if (p < 10)
log.print ("Pr. "+p+":");
else log.print ("Pr."+p+":");
log.println (formatQueue (receiveQueue[p]));
}

// キューを出力用に整形する
String formatQueue (ArrayList queue) {
if (queue.isEmpty())
return " Empty";
else {
String str = "";
for (int i=0; i<queue.size(); i++)
str += formatData (queue.get(i));
return str;
}
}


// データを出力用に整形する
String formatData (Object o) {
if (o.getClass() ==  Integer.class) {
int val = ((Integer) o).intValue();
if (val == Integer.MAX_VALUE) return " --";       // 無限大は"--"を出力
else if (val < 10) return "  "+val;
else return " "+val;
} else return " ??";
}

static void executeError (String err_mes) { /* 実行時エラー */
System.out.println("Execute error");
System.out.println(err_mes);
System.exit(1);
}
```

```
}
```

BspProcessor.java

```
import ioTools.*;  //ファイル入出力用
import java.io.*;  //ファイル入出力用

public class BspProcessor {
int processorNumber;      // プロセッサ番号
BspNetwork network;       // ネットワーク
int numOfProcessors;      // プロセッサ数
int timeI;                // 内部計算時間
int timeG;                // 通信時間
int timeL;                // 同期回数
int[] data;               // データ配列
int[] tmpData;            // 逐次ソート時の作業用データ配列

int lowProcessor;
int highProcessor;

PrintWriter output;       // 出力用ファイル
PrintWriter log;          // ログ出力用ファイル

public BspProcessor(int p, int np) {
processorNumber = p;
numOfProcessors = np;
}

// ネットワークを設定する
public void setNetwork (BspNetwork net) {
network = net;
}

// データをセットする
public void setData(int[] a) {
data = a;
}

// 時間をセット
public void setTime(int i, int g, int l) {
timeI = i;
timeG = g;
timeL = l;
}

// 出力用ファイルをセット
public void setOutputFile(PrintWriter o) {
output = o;
```

```
}

// ログ出力用ファイルをセット
public void setLogFile(PrintWriter l) {
log = l;
}


// プロセッサ low〜プロセッサ high-1 がそれぞれのデータを 2 つに分割したとき、
// それぞれの部分データの送り先のプロセッサを選ぶ
// lowProcessor に小さい部分の送り先、highProcessor に大きい部分の送り先が入る
// (前半のプロセッサ (low〜mid-1) の送り先:(low, low+1), (low+2, low+3), ...., (high-2, high-1))
// (後半のプロセッサ (mid〜high-1) の送り先:(low, low+1), (low+2, low+3), ...., (high-2, high-1))
public void selectSendingProcessors (int low, int high) {
int mid = (low+high)/2;
if (processorNumber < mid) {
lowProcessor = processorNumber*2 - low;
highProcessor = processorNumber*2 - low + 1;
} else {
lowProcessor = (processorNumber-mid)*2 + low;
highProcessor = (processorNumber-mid)*2 + low+1;
}
}

// プロセッサ low〜プロセッサ high-1 がそれぞれのデータを 2 つに分割したとき、
// それぞれの部分データの送り先のプロセッサを選ぶ
// lowProcessor に小さい部分の送り先、highProcessor に大きい部分の送り先が入る
// ただし、前半のプロセッサはプロセッサ順の通りに送り先を決定し、
// 後半のプロセッサはプロセッサ順の逆順に送り先を決定する
// (前半のプロセッサ (low〜mid-1) の送り先:(low, low+1), (low+2, low+3), ...., (high-2, high-1))
// (後半のプロセッサ (mid〜high-1) の送り先:(high-1, high-2), (high-3, high-4), ..., (low+1, low))
public void selectSendingProcessorsBitonic (int low, int high) {
int mid = (low+high)/2;
if (processorNumber < mid) {
lowProcessor = processorNumber*2 - low;
highProcessor = processorNumber*2 - low + 1;
} else {
lowProcessor = (high-1 - processorNumber)*2 + low + 1 ;
highProcessor = (high-1 - processorNumber)*2 + low;
}
}


// 前半のデータを lowProcessor に、後半のデータを highProcecceor に送る
// ただし、前半のプロセッサ (low〜mid-1) は昇順 (data[0], data[1], ..., data[length-1]) に、
// 後半のプロセッサ (mid〜high-1) は降順 (data[length-1], data[length-2], ..., data[0]) に送る
public void sendDataBitonic (int low, int high) {
```

```
int mid = (low+high)/2;
if (processorNumber < mid) {
network.putPartOfArray (lowProcessor, data, 0, data.length/2-1);
network.putPartOfArray (highProcessor, data, data.length/2, data.length-1);
} else {
network.putPartOfArrayReverce (lowProcessor, data, 0, data.length/2-1);
network.putPartOfArrayReverce (highProcessor, data, data.length/2, data.length-1);
}
}

// データを受信する
//（受信したデータは前半が昇順、後半が降順になっている）
public void receiveData () {
tmpData = network.getArray (processorNumber);
timeG += tmpData.length;
}

// データをマージする
public void mergeData() {
data = new int[tmpData.length];
int i=0;
int j=data.length-1;
for (int k=0; k<data.length; k++) // データの両端から内側に向かって走査する
if (tmpData[i] <= tmpData[j]) {
data[k] = tmpData[i];
i++;
} else {
data[k] = tmpData[j];
j--;
}
timeI += (data.length)*3;
}

// 逐次マージソート
public void sequentialMergeSort() {
if (data != null) {
tmpData = new int[data.length];
mergeSort(data, 0, data.length-1);
}
}

// 配列 a の low 番目から high 番目までをソートする
public void mergeSort (int[] a, int low, int high) {
if (high-low <= 0) {            // 範囲内の要素が 1 個以下のとき
timeI++;                    // 何もしない
} else if (high-low == 1) {    // 範囲内の要素が 2 個のとき
if (a[low] > a[high])
```

```
swap (a, low, high);   // a[low] と a[high] の値を入れ替える
timeI++;
} else {                          // 範囲内の要素が 3 個以上のとき
int mid = (low+high)/2;
timeI++;

mergeSort (a, low, mid);   // 前半部分を再帰的にソート
mergeSort (a, mid+1, high); // 後半部分を再帰的にソート

for (int i=low; i<=mid; i++) // 前半部分を tmp に昇順 (a[0]〜a[mid]) に入れる
tmpData[i] = a[i];
for (int i=mid+1; i<=high; i++) // 後半部分を tmp に降順 (a[high]〜a[mid+1]) に入れる
tmpData[mid+1+high-i] = a[i];
timeI += (high-low+1)*2;

int i=low;
int j=high;
for (int k=low; k<=high; k++) // データの両端から内側に向かって走査する
if (tmpData[i] <= tmpData[j]) {
a[k] = tmpData[i];
i++;
} else {
a[k] = tmpData[j];
j--;
}
timeI += (high-low+1)*3;
}
}

// a[i] と a[j] の値を入れ替える
void swap (int[] a, int i, int j) {
int tmp = a[i];
a[i] = a[j];
a[j] = tmp;
timeI+=3;
}

// 保持するデータ表示
public void showData() {
if (processorNumber < 10)
System.out.print("Pr. "+processorNumber+": ");
else System.out.print("Pr."+processorNumber+": ");
System.out.println (formatDataList (data));
}

// 保持するデータ出力
public void printData() {
```

```
if (processorNumber < 10)
output.print("Pr. "+processorNumber+": ");
else output.print("Pr."+processorNumber+": ");
output.println (formatDataList (data));
}

// 保持するデータ出力 (ログ出力用)
public void logData() {
if (processorNumber < 10)
log.print("Pr. "+processorNumber+": ");
else log.print("Pr."+processorNumber+": ");
log.println (formatDataList (data));
}

// データの送信先プロセッサ表示 (デバグ用)
public void showReceivers() {
if (processorNumber < 10)
System.out.print("Pr. "+processorNumber+": ");
else System.out.print("Pr."+processorNumber+": ");
System.out.println ("lowProcessor ="+formatData (lowProcessor)+" highProcessor ="+formatData (highProcesso
}

// データの送信先プロセッサ出力 (ログ出力用)
public void logReceivers() {
if (processorNumber < 10)
log.print("Pr. "+processorNumber+": ");
else log.print("Pr."+processorNumber+": ");
log.println ("lowProcessor ="+formatData (lowProcessor)+" highProcessor ="+formatData (highProcessor));
}

// 出力用にデータ配列を整形
String formatDataList (int[] d) {
if (d == null)
return "Empty";
else {
String str = "";
for (int i=0; i<d.length; i++)
str += formatData (d[i]);
return str;
}
}

// 出力用にデータを整形
String formatData (int d) {
if (d < 10) return "  " + d;
else return " " + d;
}
```

```
}
```

InputData.java

```
import ioTools.*;           // ファイル入出力用
import java.io.*;           // ファイル入出力用
import java.util.*;         // 文字列処理用

public class InputData {
final int range = 100;                  // データの範囲
int[] array;                            // データ

public InputData (int n) {
array = new int [n];
}

// 要素数 array.length の 0〜range-1 までのランダムデータを作成する
public void makeRandomData () {
for (int i=0; i<array.length; i++)
array[i] = (int) (Math.random() * range);
}

// 要素数 array.length のソート済みデータを作成する
public void makeSortedData () {
for (int i=0; i<array.length; i++)
array[i] = (int) (i*range/array.length);
}

// 全てのデータを得る
public int[] get () {
return array;
}

// array[low]〜array[high] までのデータを得る
public int[] get (int low, int high) {
if (low < 0 || high >= array.length || low > high)
return null;
else {
int[] a = new int[high - low +1];
for (int i=0; i<a.length; i++)
a[i] = array[low + i];
return a;
}
}

// データを表示する
public void dump () {
for (int i=0; i<array.length; i++)
System.out.print (array[i] + " ");
```

```
System.out.println();
}

// データをデフォルトファイルに出力する
public void dumpToFile () {
dumpToFile ("InputData.txt");
}

// データを指定したファイルに出力する
public void dumpToFile (String fileName) {
PrintWriter outputFile = FileIo.fWrite(fileName, false);
for (int i=0; i<array.length; i++)
outputFile.print (array[i] + " ");
outputFile.println();
outputFile.close();
}

// デフォルトファイルからデータを読み込む
public void readData () {
readData ("InputData.txt");
}

// 指定したファイルからデータを読み込む
public void readData(String fileName) {
BufferedReader buffer = FileIo.fRead(fileName);
String line = readLine(buffer);
StringTokenizer st = new StringTokenizer(line);
int arraySize = st.countTokens();
for (int i=0; i<arraySize; i++)
array[i] = Integer.parseInt(st.nextToken());
}

// ファイルから 1 行読み込む
String readLine(BufferedReader buffer) {
String line = "";
try {
line = buffer.readLine();
} catch(IOException error_report) {
            /* 読込みエラーが発生したら，キャッチした例外を表示し，
               ファイルを閉じ，処理系を終了させる */
System.out.println(error_report);
System.exit(1);
}
return line;
}

public static void main (String[] args) {
```

```
System.out.print("size : ");
int n = Console.ReadInteger();
InputData data = new InputData(n);
data.makeRandomData();
data.dump ();
data.dumpToFile();
}
}
```

Garph.java

```
import java.io.*;
import java.awt.*;
import java.awt.event.*;

public class Graph extends Frame {

int max, min, step;   // データの最大値, 最小値, 目盛幅
int axis = 0;   // =0 : 表の通り,  =1 : 表の列と行を逆にしたグラフ
int line = 0;   // =0 : 棒グラフ,  =1 : 折れ線グラフ
int row, col;   // 表の行数と列数
int width = 1000, height = 600;   // Window の大きさ
int x_l = 10, x_r = 150, y_u = 40, y_d = 40;   // 左右上下の余白
int f_b_s = 20, f_s_s = 16;   // 大, 小のフォントサイズ
String title;
String [][] table;
String [][] table_a;
String [][] table_b;
Font f_b, f_bp, f_s, f_sp;

/*********************************************************************/
/*     コンストラクタ                                                */
/*          title_i : グラフのタイトル                               */
/*          tabel_i : グラフの元データ                               */
/*                       1行目：横軸目盛への表示項目                 */
/*                       1列目：凡例名                               */
/*          max_i, min_i, step_i : データの最大値, 最小値, 目盛幅 */
/*********************************************************************/
Graph(String title_i, String [][] table_i, int max_i, int min_i, int step_i)
{
// Frame クラスのコンストラクタの呼び出し
super("グラフ");
// フォントの設定
f_b  = new Font("TimesRoman", Font.BOLD, f_b_s);
f_bp = new Font("TimesRoman", Font.PLAIN, f_b_s);
f_s  = new Font("TimesRoman", Font.BOLD, f_s_s);
f_sp = new Font("TimesRoman", Font.PLAIN, f_s_s);
// テーブルデータの保存
int i1, i2;

row   = table_i.length;
col   = table_i[0].length;
title = title_i;

table_a = new String [row][col];
for (i1 = 0; i1 < row; i1++) {
```

```
for (i2 = 0; i2 < col; i2++)
table_a[i1][i2] = table_i[i1][i2];
}

table_b = new String [col][row];
for (i1 = 0; i1 < col; i1++) {
for (i2 = 0; i2 < row; i2++)
table_b[i1][i2] = table_i[i2][i1];
}

table = table_a;
max   = max_i;
min   = min_i;
step  = step_i;
// Window サイズと表示位置を設定
setSize(width, height);
Toolkit tool = getToolkit();
Dimension d = tool.getScreenSize();
setLocation(d.width / 2 - width / 2, d.height / 2 - height / 2);
// ウィンドウを表示
setVisible(true);
// イベントアダプタ
addWindowListener(new WinEnd());
addMouseListener(new ClickMouse());
addComponentListener(new ComponentResize());
}

/*************************************/
/*     与えられたデータよりグラフを作成 */
/*          coded by Y.Suganuma         */
/*************************************/

/************/
/*     描画 */
/************/
public void paint (Graphics g)
{
int i1, i2, k, len, x1, x2, y1, y2, sp, x_b1, x_b2, y_b1, y_b2, w;
FontMetrics fm;
// Window サイズの取得
Dimension d = getSize();
width  = d.width;
height = d.height;
// 軸変更ボタン（左上，幅，高さ）
g.setColor(Color.yellow);
g.fill3DRect(width - 3 * f_b_s - 10, y_u, 3 * f_b_s, f_b_s + f_b_s / 2, true);
g.setColor(Color.black);
```

```
g.setFont(f_bp);
g.drawString("縦/横", width - 3 * f_b_s - 10 + f_b_s / 3, y_u + f_b_s + f_b_s / 7);
// グラフ変更ボタン（左上，幅，高さ）
g.setColor(Color.pink);
g.fill3DRect(width - 3 * f_b_s - 10, y_u + 2 * f_b_s, 3 * f_b_s, f_b_s + f_b_s / 2, true);
g.setColor(Color.black);
g.drawString("棒/線", width - 3 * f_b_s - 10 + f_b_s / 3, y_u + 3 * f_b_s + f_b_s / 7);
// 表の表示ボタン（左上，幅，高さ）
g.setColor(Color.orange);
g.fill3DRect(width - 3 * f_b_s - 10, y_u + 4 * f_b_s, 3 * f_b_s, f_b_s + f_b_s / 2, true);
g.setColor(Color.black);
g.drawString("  表", width - 3 * f_b_s - 10 + f_b_s / 3, y_u + 5 * f_b_s + f_b_s / 7);
// グラフタイトルの表示
g.setFont(f_b);
fm  = g.getFontMetrics(f_b);
len = fm.stringWidth(title);
g.drawString(title, (width - x_r) / 2 - len / 2, y_u + f_b_s);
// ｘ軸とｙ軸
fm = g.getFontMetrics(f_s);
x1 = fm.stringWidth(Integer.toString(max));
y1 = fm.stringWidth(Integer.toString(min));
if (y1 > x1)
x1 = y1;
x_b1 = x_l + x1 + f_s_s / 5;
x_b2 = width - x_r - f_s_s / 2;
y_b1 = height - y_d - f_s_s - f_s_s / 5;
y_b2 = y_u + f_b_s + f_s_s / 2;
g.drawLine(x_b1, y_b1-1, x_b2, y_b1-1);
g.drawLine(x_b1, y_b1, x_b2, y_b1);
g.drawLine(x_b1, y_b1+1, x_b2, y_b1+1);
g.drawLine(x_b1-1, y_b1, x_b1-1, y_b2);
g.drawLine(x_b1, y_b1, x_b1, y_b2);
g.drawLine(x_b1+1, y_b1, x_b1+1, y_b2);
// ｘ軸の目盛と目盛り線
g.setFont(f_sp);
sp = (x_b2 - x_b1) / (2 * (col - 1));
x1 = x_b1 + sp;
y1 = height - y_d;
for (i1 = 1; i1 < col; i1++) {
len = fm.stringWidth(table[0][i1]);
g.drawString(table[0][i1], x1 - len / 2, y1);
x1 += 2 * sp;
}
// ｙ軸の目盛と目盛り線
sp = (max - min) / step;
sp = (y_b1 - y_b2) / sp;
x1 = min;
```

```
y1 = y_b1;
g.drawString(Integer.toString(x1), x_l, y1 + f_s_s / 2);
while (x1 < max) {
x1 += step;
y1 -= sp;
g.drawLine(x_b1, y1, x_b2, y1);
g.drawString(Integer.toString(x1), x_l, y1 + f_s_s / 2);
}
// 凡例
Color cl[] = new Color [9];
cl[0] = Color.magenta;
cl[1] = Color.blue;
cl[2] = Color.orange;
cl[3] = Color.cyan;
cl[4] = Color.pink;
cl[5] = Color.green;
cl[6] = Color.yellow;
cl[7] = Color.darkGray;
cl[8] = Color.red;

y1 = y_u + 7 * f_b_s;
for (i1 = 1; i1 < row; i1++) {
k = (i1 - 1) % 9;
g.setColor(cl[k]);
g.fillRect(width - x_r, y1, 15, 5);
g.drawString(table[i1][0], width - x_r + 20, y1 + f_s_s / 2);
y1 += (f_s_s + 5);
}
// グラフの作図（棒グラフ）
sp = (x_b2 - x_b1) / (2 * (col - 1));
w  = (x_b2 - x_b1) / (row * (col - 1));
if (line == 0) {
for (i1 = 1; i1 < row; i1++) {
x1 = x_b1 + sp;
k = (i1 - 1) % 9;
g.setColor(cl[k]);
for (i2 = 1; i2 < col; i2++) {
x2 = x1 - w * (row - 1) / 2 + w * (i1 - 1);
y2 = (int)(y_b1 - (y_b1 - y_b2) *
                        (Double.parseDouble(table[i1][i2]) - min) / (max - min));
g.fillRect(x2, y2, w, y_b1 - y2);
x1 += 2 * sp;
}
}
}
// グラフの作図（折れ線グラフ）
else {
```

```
for (i1 = 1; i1 < row; i1++) {
x2 = x_b1 + sp;
k  = (i1 - 1) % 9;
g.setColor(cl[k]);
for (i2 = 1; i2 < col; i2++) {
y2 = (int)(y_b1 - (y_b1 - y_b2) *
                          (Double.parseDouble(table[i1][i2]) - min) / (max - min));
g.fillOval(x2-2, y2-2, 6, 6);
if (i2 > 1)
g.drawLine(x1, y1, x2, y2);
x1 = x2;
y1 = y2;
x2 += 2 * sp;
}
}
}
}

/***************************************/
/*     マウスがクリックされたときの処理 */
/***************************************/
class ClickMouse extends MouseAdapter
{
public void mouseClicked(MouseEvent e)
{
int xp, yp, x1, x2, y1, y2;
// マウスの位置
xp = e.getX();
yp = e.getY();

x1 = width - 3 * f_b_s - 10;
y1 = y_u;
x2 = x1 + 3 * f_b_s;
y2 = y1 + f_b_s + f_b_s / 2;
// ｘ軸とｙ軸を交換
if (xp > x1 && xp < x2 && yp > y1 && yp < y2) {
if (axis == 0) {
table = table_b;
axis  = 1;
}
else {
table = table_a;
axis  = 0;
}
row = table.length;
col = table[0].length;
repaint();
```

```
}
else {
x1 = width - 3 * f_b_s - 10;
y1 = y_u + 2 * f_b_s;
x2 = x1 + 3 * f_b_s;
y2 = y1 + f_b_s + f_b_s / 2;
// 棒グラフと折れ線グラフ
if (xp > x1 && xp < x2 && yp > y1 && yp < y2) {
if (line == 0)
line = 1;
else
line = 0;
repaint();
}
else {
x1 = width - 3 * f_b_s - 10;
x2 = x1 + 3 * f_b_s;
y1 = y_u + 4 * f_b_s;
y2 = y1 + f_b_s + f_b_s / 2;
// 表の表示
if (xp > x1 && xp < x2 && yp > y1 && yp < y2) {
setVisible(false);
Table tb = new Table(title, table, max, min, step);
}
}
}
}
}

/*************************/
/*     Window のサイズ変化 */
/*************************/
class ComponentResize extends ComponentAdapter
{
public void componentResized(ComponentEvent e)
{
repaint();
}
}

/****************/
/*     終了処理 */
/****************/
class WinEnd extends WindowAdapter
{
public void windowClosing(WindowEvent e) {
setVisible(false);
```

```
System.exit(1);
}
}
}
```

DrawGraph.java

```
//グラフの描画
import java.awt.*;
class DrawGraph {
   private int Xo = 0, //中心座標(x,y)
        Yo = 0,
                  r = 0, //半径
l,//データ数
deg[]; //角度

   private double gr[],//データ用配列
   sum = 0;

   private String name[]; //ラベル用


//コンストラクタ3種類
   public DrawGraph ( int x, int y, int ra, double a[])
   {
sum = 0;
Xo = x;
Yo = y;
r = ra;
l = a.length;
deg = new int[l];
gr = new double[l];
for( int  i = 0;  i <= l - 1 ;i ++)
{
   gr[i] = a[i];
   sum += gr[i];
}
for( int  i=0;  i <= l - 1 ;i ++)
{
   deg[i] = (int)(Math.round(gr[i]/sum* 360));
}
rarrngement();
   }

   public DrawGraph ( int x, int y , int ra, int a[])
   {
sum = 0;
Xo = x;
Yo = y;
r = ra;
l = a.length;
deg = new int[l];
```

```
gr = new double[l];
for( int  i = 0;  i <= l - 1 ;i ++)
{
   gr[i] = a[i];
   sum += gr[i];
}
for( int  i=0;  i <= l - 1 ;i ++)
{
   deg[i] = (int)(Math.round(gr[i]/sum* 360));
          }
rarrngement();
   }

   public DrawGraph ( int x, int y , int ra, double a[],String Name[])
   {
sum = 0;
Xo = x;
Yo = y;
r = ra;
l = a.length;
deg = new int[l];
gr = new double[l];
name = new String[l];
for( int  i = 0;  i <= l - 1 ;i ++)
{
   gr[i] = a[i];
   sum += gr[i];
   name[i] = Name[i];
}
for( int  i=0;  i <= l - 1 ;i ++)
{
   deg[i] = (int)(Math.round(gr[i]/sum* 360));
}
rarrngement();
   }


  public void paint (Graphics g){

int now =90;
for( int t =0; t <= l - 1; t++)
{
   g.setColor(new Color((t * 50) % 255,Math.abs((255- t * 80)) % 255,(t*t+2*t) % 255));  //色の設定

   g.fillArc(Xo-r,Yo-r,2*r,2*r,now,-deg[t]); //扇形の描画
             now -= deg[t];
}
```

```
g.setColor(Color.black);
int Sum = 0;
for( int p =0; p <= l - 1 ; p++){
   g.drawLine(Xo,Yo,Xo+(int)(r*Math.sin(Math.PI*2*(Sum+deg[p])/360)),Yo+(int)(-r*Math.cos(Math.PI*2*(Sum+d
//区切り線を描画
   Sum += deg[p];
}

 g.drawOval(Xo-r,Yo-r,2*r,2*r); //外円の描画
   }

//ラベルと色リストを表示
   public void DrawLabel (Graphics g, int x, int y, int n, int w, boolean ori){
if(ori == true){ //縦に n 個ずつ幅 w で並べる。
   int col = l / n;
   int t = 0;
   for( int c = 0; c <= col + 1; c++){
for( int r = 0; r < n && t < l; r++){
   g.setColor(new Color((t * 50) % 255,Math.abs((255 - t * 80)) % 255,(t*t+2*t) % 255));
   g.fillRect(x + w * c , y + 15 * r , 16, 12);
            g.setColor(Color.black);
  g.drawString(name[t],x + w * c + 18, y + 15 * r + 10);
  t++;
}
   }
}
else {   //横に n 個ずつ幅 w で並べる。
   int row= l / n;
   int t = 0;
   for( int r = 0; r <= row + 1; r++){
for( int c = 0; c < n && t < l; c++){
   g.setColor(new Color((t * 50) % 255,Math.abs((255 - t * 80)) % 255,(t*t+2*t) % 255));
   g.fillRect(x + w * c , y + 15 * r , 16, 12);
            g.setColor(Color.black);
  g.drawString(name[t],x + w * c + 18, y + 15 * r + 10);
  t++;
}
   }
}
}

//円の外にラベルを表示
   public void DrawLabel2 (Graphics g){
int Sum = 0;
for( int p =0; p <= l - 1 ; p++){
   double flen = 15 + (12 * (p % 5));
   g.drawString(name[p],Xo+(int)((r+40)*Math.sin(Math.PI*2*(Sum+deg[p]/2)/360)),Yo+(int)(-(r+flen)*Math.co
```

```
   Sum += deg[p];
}
    }

   public void rarrngement(){
       for( int d = 0; d < l; d++){
      if ( deg[d] == 0){
 l--;
if (d < l){
             for( int hj = d; hj < l; hj++){
      name[hj] = name[hj+1];
      deg[hj] = deg[hj + 1];
        }
    d--;
}
      }
   }
    }
}
```

Table.java

```
/**********************************/
/*     与えられたデータより表を作成 */
/*          coded by Y.Suganuma     */
/**********************************/
import java.io.*;
import java.awt.*;
import java.awt.event.*;

public class Table extends Frame {

int max, min, step;   // データの最大値, 最小値, 目盛幅
int row, col;   // 表の行数と列数
int width = 900, height = 600;   // Window の大きさ
int x_l = 10, x_r = 10, y_u = 40, y_d = 40;   // 左右上下の余白
int f_b_s = 20, f_s_s = 16;   // 大, 小のフォントサイズ
String title;
String [][] table;
String [][] table_a;
String [][] table_b;
Font f_b, f_bp, f_s, f_sp;

/*********************************************************************/
/*     コンストラクタ                                                */
/*          title_i : グラフのタイトル                               */
/*          tabel_i : グラフの元データ                               */
/*                       1行目：横軸目盛への表示項目                 */
/*                       1列目：凡例名                               */
/*          max_i, min_i, step_i : データの最大値, 最小値, 目盛幅 */
/*********************************************************************/
Table(String title_i, String [][] table_i, int max_i, int min_i, int step_i)
{
// Frame クラスのコンストラクタの呼び出し
super("表");
// フォントの設定
f_b  = new Font("TimesRoman", Font.BOLD, f_b_s);
f_bp = new Font("TimesRoman", Font.PLAIN, f_b_s);
f_s  = new Font("TimesRoman", Font.BOLD, f_s_s);
f_sp = new Font("TimesRoman", Font.PLAIN, f_s_s);
// テーブルデータの保存
int i1, i2;
double value;

row   = table_i.length;
col   = table_i[0].length;
title = title_i;
```

```
table_a = new String [row][col];
for (i1 = 0; i1 < row; i1++) {
for (i2 = 0; i2 < col; i2++)
table_a[i1][i2] = table_i[i1][i2];
}

table_b = new String [col][row];
for (i1 = 0; i1 < col; i1++) {
for (i2 = 0; i2 < row; i2++)
table_b[i1][i2] = table_i[i2][i1];
}

table = table_a;
max   = max_i;
min   = min_i;
step  = step_i;
// Window サイズと表示位置を設定
setSize(width, height);
Toolkit tool = getToolkit();
Dimension d = tool.getScreenSize();
setLocation(d.width / 2 - width / 2, d.height / 2 - height / 2);
// ウィンドウを表示
setVisible(true);
// イベントアダプタ
addWindowListener(new WinEnd());
addMouseListener(new ClickMouse());
addComponentListener(new ComponentResize());
}

/************/
/*     描画 */
/************/
public void paint (Graphics g)
{
int i1, i2, sp1, sp2, len, x1, x_b1, x_b2, y1, y_b1, y_b2;
FontMetrics fm;
// Window サイズの取得
Dimension d = getSize();
width  = d.width;
height = d.height;
// 軸変更ボタン（左上, 幅, 高さ）
g.setColor(Color.orange);
g.fill3DRect(width - 4 * f_b_s - 10, y_u, 4 * f_b_s, f_b_s + f_b_s / 2, true);
g.setColor(Color.black);
g.setFont(f_bp);
g.drawString("グラフ", width - 4 * f_b_s - 10 + f_b_s / 2, y_u + f_b_s + f_b_s / 7);
```

```
// 表タイトルの表示
g.setFont(f_b);
fm  = g.getFontMetrics(f_b);
len = fm.stringWidth(title);
g.drawString(title, (width - x_r) / 2 - len / 2, y_u + f_b_s);
// 罫線の表示
x_b1 = x_l;
x_b2 = width - x_r;
y_b1 = y_u + f_b_s + f_s_s / 2;
y_b2 = height - y_d;

sp1  = (y_b2 - y_b1) / row;
y1   = y_b1;
for (i1 = 0; i1 <= row; i1++) {
g.drawLine(x_b1, y1, x_b2, y1);
y1 += sp1;
}

sp2  = (x_b2 - x_b1) / col;
x1   = x_b1;
for (i1 = 0; i1 <= col; i1++) {
g.drawLine(x1, y_b1, x1, y_b2);
x1 += sp2;
}
// 表の列および行の見出しの表示
g.setFont(f_s);
fm = g.getFontMetrics(f_s);

y1 = y_b1 + sp1 + sp1 / 2 + f_s_s / 2;
for (i1 = 1; i1 < row; i1++) {
len = fm.stringWidth(table[i1][0]);
x1  = x_b1 + sp2 / 2 - len / 2;
g.drawString(table[i1][0], x1, y1);
y1 += sp1;
}

x1 = x_b1 + sp2 + sp2 / 2;
y1 = y_b1 + sp1 / 2 + f_s_s / 2;
for (i1 = 1; i1 < col; i1++) {
len = fm.stringWidth(table[0][i1]);
g.drawString(table[0][i1], x1 - len / 2, y1);
x1 += sp2;
}
// 表データの表示
g.setFont(f_sp);
fm = g.getFontMetrics(f_sp);
```

```
y1 = y_b1 + sp1 + sp1 / 2 + f_s_s / 2;
for (i1 = 1; i1 < row; i1++) {
x1 = x_b1 + 2 * sp2;
for (i2 = 1; i2 < col; i2++) {
len = fm.stringWidth(table[i1][i2]);
g.drawString(table[i1][i2], x1 - len - 10, y1);
x1 += sp2;
}
y1 += sp1;
}
}

/*************************************/
/*     マウスがクリックされたときの処理 */
/*************************************/
class ClickMouse extends MouseAdapter
{
public void mouseClicked(MouseEvent e)
{
int xp, yp, x1, x2, y1, y2;
// マウスの位置
xp = e.getX();
yp = e.getY();
// グラフの表示
x1 = width - 4 * f_b_s - 10;
y1 = y_u;
x2 = x1 + 4 * f_b_s;
y2 = y1 + f_b_s + f_b_s / 2;
if (xp > x1 && xp < x2 && yp > y1 && yp < y2) {
setVisible(false);
Graph tb = new Graph(title, table, max, min, step);
}
}
}

/*************************/
/*     Window のサイズ変化 */
/*************************/
class ComponentResize extends ComponentAdapter
{
public void componentResized(ComponentEvent e)
{
repaint();
}
}

/***************/
```

```
/*     終了処理 */
/****************/
class WinEnd extends WindowAdapter
{
public void windowClosing(WindowEvent e) {
setVisible(false);
System.exit(1);
}
}
}
```

Hikakup1g.java

```
public class Hikakup1g {


public static void main(String args[]){

int n=1;
    int N =14; //表示させたいデータの数

String [][] tab = new String [100][100];


double [][] YY = new double [100][100];
double [] XX = new double [100];
double c=5;
int ii=0;
while(n<N){



double mm=0;
double m=1;
double i=0;
double b=0;
while(i<N)
{
if(i==0){
 mm=(int)Math.pow(2,n+1);

}else{
mm  += 2*(m+1)*(int)Math.pow(2,n-1);
}
b+=Math.pow(2,m+n-1)*(m+2+n-1);
double nn =  (int)Math.pow(2,n);
double mmm=mm*c/nn;

double bb=b-mmm;
double bbb= (bb/mm);
if(i>1){
 bbb+=  +2;
}
if(bb<0){
bbb=0;
}
tab[n-1][(int)i]=Integer.toString((int)bbb);
YY[n-1][(int)i]=bbb;
```

```
double p=(int)Math.pow(2,i+1);
XX[(int)i]=p;
m++;
i++;
}
c+=Math.pow(2,n)*(n+3);

n++;
ii=(int)i;
}


  String [][] table = new String [n][ii];




  int j2=0;
  int i3=1;
  while(j2<ii-1){
  int i2=0;

  while(i2+i3-1<n-1){

  if(tab[i2][j2]==null)
{
tab[i2][j2]="0";
}
  table[i3+i2][1+j2]= tab[i2][j2];
  i2++;

  table[i2][0]="n"+Integer.toString((int)Math.pow(2,i2+1));

  int i4=0;
  while(i4<n-1){
  if(table[1+i4][1+j2]==null)
{
table[1+i4][1+j2]="0";
}
  i4++;
  }




  }
  j2++;
```

```
 i3++;
  table[0][j2]="p "+Integer.toString((int)XX[j2-1]);
  }


int iiii=0;
double wari=0;

int jou2=0;
while(n>iiii)
{
int gg = 0;
while(ii>gg){
int jou=1;
wari = YY[iiii][gg];

while(1<wari){
wari = wari/10;
jou = jou *10;

}

if(jou>jou2)
{
jou2=jou;
}

gg++;
}
iiii++;
}




  String title ="プロセッサ１台を超える計算量の時の g の値";

  Graph gp= new Graph(title, table, jou2, 0,jou2/10);//グラフの表示




}
```

```
}
```

```
Hikakup1L.java

public class Hikakup1L {
public static void main(String args[])
{

double n=  Math.pow(2,1) ;   // データ数初期値
double p=1;   // プロセッサ数
int L=1;   // 同期時間
int g=3;  //通信時間
    int mn=10; //表示させるデータ数



int dais=2; // １台の計算量を越える台数

int daisuu=0;
while(dais!=1){
daisuu++;
dais/=2;
}
double y= 0;
 double yyy=0;

int i=0;
int gi=1;

double [][] YY = new double [1000][700];
String [][][] tab = new String [5][1000][700];

double [][] ZZ = new double [1000][700];
while(n <= Math.pow(2,mn) ){ //mn まで n をたす
double p2=p;


double count =1;
double logd=0 ;
double ap =0;
double cp = 0;
double logp =0;
int ii=0;
while(n/p2 > 1 ){
double d=n/p2;


  double lo=0;
double dd=d;
```

```
while(dd!=1){




lo++;
logd = lo;
dd/=2;

}


double lop=0;

double pp=p2;

while(pp!=1){




lop++;
logp = lop;
pp/=2;

}
     if(ii>0){
 ap = 1+logp;
}
else if(ii==0){
ap =0;
}
double bp = cp + ap;

 cp = bp;

double v =(d * logd * (bp+1));//内部計算
double x = (d*bp*g);   //通信時間
double w =(bp*L);//同期時間

 y =v+x+w;//総合計算量

YY[ii][gi-1] = y;
ZZ[ii][gi-1]=bp;
tab[0][ii][gi-1]  = Integer.toString((int)YY[ii][gi-1]);
```

```
tab[1][ii][gi-1] = Double.toString(p2);



System.out.println(bp+""+"プロセッサ台数 " + p2 +" 同期時間 " + w + " 通信時間 " + x + " 総合計算時間 " + y);

yyy=y;


if(ii==0){
yyy=y;

}
 else if(ii>0){
if(yyy>y){
yyy= y;
count  = Math.pow(2, ii);

}

}

System.out.println(YY[daisuu][gi-1]/YY[0][gi-1]);

if(YY[0][gi-1]<=YY[daisuu][gi-1]){
System.out.println(d);}

 if(n==1){System.out.println(" 最速計算時間 "+yyy);
  System.out.println( " 最速の台数 " + count);
 }
p2 = p2*2;
 ii++;


}

n*= 2;
gi++;
i= ii;
}

String [][] table = new String [gi][i+1];
String [][] table2 = new String [gi][i+1];
String title ="g="+g+ "の時各 p が p1 を超える L の値";

int ten =0;
int ten2 =0;
```

```
while(ten2<gi){
while(ten<i+1){


table[ten2][ten]="";
ten++;
}
ten2++;
}
int gken=0;
int dat =1;



while(gken<gi-1){
int ken =0;
while(ken<i){

if(tab[0][ken][gken]==null)
{
tab[0][ken][gken]="0";
}


    table[1+gken][ken+1]= tab[0][ken][gken];
    if(gken==0){
    table[1+gken][0]="n"+Integer.toString(2*dat);
    }
    else{
     table[1+gken][0]=Integer.toString(2*dat);

    }
    table[1+gken][0]  = table[1+gken][0].concat("");

   int yz= (int)((YY[0][gken]-YY[ken][gken])/ZZ[ken][gken]);
   if(yz<=10000000 && yz >=0){
    table2[1+gken][ken+1]=Integer.toString(yz+2);
   }
   else if(yz<=0){

   table2[1+gken][ken+1]="0";
   }
   if(gken==0){
   table2[1+gken][0]="n"+Integer.toString(2*dat);
   }
```

```
   else {
   table2[1+gken][0]=Integer.toString(2*dat);
   }
   table2[1+gken][0]  = table[1+gken][0].concat("");

    if(1+ken==1){
table2[1+gken][ken+1]="0";
}



    if(table2[1+gken][ken+1]==null)
{
table2[1+gken][ken+1]="0";
}
    ken++;

    }
dat*=2;
gken++;
}
```

```
int  Lken =0;
while(Lken<i){
if(Lken==0){
table[0][1+Lken]="p"+Integer.toString((int)Math.pow(2,Lken));
}
else{
table[0][1+Lken]=Integer.toString((int)Math.pow(2,Lken));

}

table[0][1+Lken]= table[0][1+Lken].concat("");
table2[0][1+Lken]=Double.toString(Math.pow(2,Lken));
table2[0][1+Lken]= table[0][1+Lken].concat("");
```

```
Lken++;
}




int iiii=0;
double wari=0;

int jou2=0;
while(i>iiii)
{
int gg = 0;
while(gi>gg){
double jou=1;
wari = YY[iiii][gg];
while(1<wari){
wari = wari/10;
jou = jou *10;

}

if(jou>jou2)
{
jou2=(int)jou;

}

gg++;
}
iiii++;
}



int rr =0;
int rrr =1;
int wari2 =3;

//wari2 初期値 3
while(rr<i){
int r =0;
while(r<gi){
if((int)YY[rrr][r]!=0){
System.out.println((int)YY[rr][r]+"  "+wari2+"  " + ((int)YY[rr][r] -(int)YY[rrr][r]) + "   " +    ((int)Y
}
r++;
```

```
}
wari2 += 1;
rr++;
rrr++;
System.out.println("   ");
}



Graph gp= new Graph(title, table2, jou2, 0,jou2/10); //グラフの表示

}




}
```

Hikakup2pL.java

```
public class Hikakup2pL {


public static void main(String args[]){



String [][][] tab = new String [100][70][100];


double [][][] YY = new double [100][70][100];


int g=1; //  通信時間
int G=g+(3);//g で設定した値から、順に１づつ g を足した数を表示させたいだけ () の中に入れる
int gi=1;
while(g<G){
int o=-4*g;
double o4;
double o3=0;
double o2;
int M=2;
int I=10;//データ数８以降の表示させたい数だけ設定する
int J=I;
int hi=0 ;
int jj=0;



double ii=0;


double Dm=0;
while(ii<I){
int gg=0;

int wbp=3;
System.out.println(-o);
o4=(int)Math.pow(2,ii+1);
o2 =ii+3;
o3= o4*o2;
double Am=0;
```

```
double Bm=0;
double Cm=0;
double Am2=1;
double Cm2=0;
int mm=1;
while(mm<M){
 Am=Math.pow(2,mm);
Am2+=Am;
Bm=(3+g)*Math.pow(2,mm);

if(mm>1){
  Cm=mm*Math.pow(2,mm-1);


}
Cm2+=Cm;
mm++;


}


int j=0;

Dm= Am2+Bm+Cm2;

int ocp=-o;
while(j<J){
ocp-=Dm;
tab[(int)ii][j][gi-1]=Integer.toString((int)(-ocp/wbp)+2);
YY[(int)ii][j][gi-1]=ocp;
Dm+=Bm+Cm2;

j++;
wbp++;
}

o-=o3;
ii++;
M++;
jj=j;
if(g>1){
gg= 2*(g-1)*(int)Math.pow(2,ii);


}
```

```
o-=gg;
}




hi=(int)ii;


  String [][] table = new String [hi+1][jj+1];


int ten =0;
int ten2 =0;


while(ten2<hi){
while(ten<jj){


table[ten2][ten]="";
ten++;
}
ten2++;
}



  int j2=0;
  int i3=1;
  while(j2<jj){
  int i2=0;

  while(i3+i2-1<hi){

  if(tab[i2][j2][gi-1]==null)
{
tab[i2][j2][gi-1]="";
}
  table[i3+i2][1+j2]= tab[i2][j2][gi-1];
  int i4=0;
  while(i4<hi){
  if(table[1+i4][1+j2]==null)
{
table[1+i4][1+j2]="0";
}
```

```
  i4++;
  }
  i2++;
  if(i2==1){
  table[i2][0]="n"+Integer.toString((int)Math.pow(2,i2)*4);
  }
  else{

  table[i2][0]=Integer.toString((int)Math.pow(2,i2)*4);//データ数の値をグラフのパラメタに入れる
  }

  }
i3++;
  j2++;
  if(j2==1){
  table[0][j2]="p"+Integer.toString((int)Math.pow(2,j2)*2);
  }
  else{

  table[0][j2]=Integer.toString((int)Math.pow(2,j2)*2);//プロセッサの値をグラフのパラメタに入れる
  }

  }
```

```
int iiii=0;
double wari=0;

int jou2=0;
while(hi>iiii)
{
int gg = 0;
while(jj>gg){
int jou=1;
wari = -YY[iiii][gg][gi-1];
while(1<wari){
wari = wari/10;
```

```
jou = jou *10;

}

if(jou>jou2)
{
jou2=jou;
}

gg++;
}
iiii++;
}









  String title ="g="+g+" p が 1/2 倍の p を超える L の値  ";
Graph gp= new Graph(title, table, jou2/1000, 0,jou2/10000);//グラフの表示


 g++;
}







}
}
```

Testpp.java

```
import java.io.*;

public class Testpp {
public static void main(String args[])
{

double n=  Math.pow(2,1) ;   // データ数
double p=1;   // プロセッサ数
int L=1;   // 同期時間
int g=1;  //通信時間

boolean gl =false;

double bain=n;

double baigg = 10*bain;

int dais=2; // １台の計算量を越える台数

int daisuu=0;
while(dais!=1){
daisuu++;
dais/=2;
}
double y= 0;
 double yy = 0;
 double yyy=0;

int i=0;
int gi=1;

int mn=9;

double [][] YY = new double [1000][700];
String [][][] tab = new String [5][1000][700];
while(n <= Math.pow(2,mn) ){
double p2=p;


double count =1;
double logd=0 ;
double ap =0;
double cp = 0;
double logp =0;
int ii=0;
```

```
while(n/p2 > 1 ){
double d=n/p2;


  double lo=0;
double dd=d;
while(dd!=1){




lo++;
logd = lo;
dd/=2;

}


double lop=0;

double pp=p2;

while(pp!=1){




lop++;
logp = lop;
pp/=2;

}
     if(ii>0){
 ap = 1+logp;
}
else if(ii==0){
ap =0;
}
double bp = cp + ap;

 cp = bp;

double logdd = Math.log(d);
double v =(d * logd * (bp+1));
double x = (d*bp*g);
```

```
double w =(bp*L);
double z = (bp * d);
double a = v+z;

 y =v+x+w;

YY[ii][gi-1] = y;

tab[0][ii][gi-1]  = Integer.toString((int)YY[ii][gi-1]);
tab[1][ii][gi-1] = Double.toString(p2);



System.out.println(ii+" "+v+""+"プロセッサ台数 " + p2 +" 同期時間 " + w + " 通信時間 " + x + " 総合計
算時間 " + y);

yyy=y;


//   Graph gp= new Graph(タイトル, データ, y軸目盛り最大値,
//                         y軸目盛り最小値, y軸目盛線の幅);
// 以下のように書けば, 表が表示される
//   Table tb= new Table(title, table, 100, 0, 10);

if(ii==0){
yyy=y;

}
 else if(ii>0){
if(yyy>y){
yyy= y;
count  = Math.pow(2, ii);

}

}

System.out.println(YY[daisuu][gi-1]/YY[0][gi-1]);

if(YY[0][gi-1]<=YY[daisuu][gi-1]){
System.out.println(d);}

 if(n==1){System.out.println(" 最速計算時間 "+yyy);
  System.out.println( " 最速の台数 " + count);
 }
p2 = p2*2;
 ii++;
```

```
}

n*= 2;
gi++;
i= ii;
}

String [][] table = new String [gi][i+1];
String title ="g="+g +"L="+ L +"シミュレータ";
String XX="";

int ben =1;
int ten =0;
int ten2 =0;


while(ten2<gi){
while(ten<i+1){


table[ten2][ten]="";
ten++;
}
ten2++;
}
int gken=0;
int dat =1;


//gi=1 なら ii=2 gi=2 nara ii=3


while(gken<gi-1){
int ken =0;
int iken =gken;
while(ken<i){

if(tab[0][ken][gken]==null)
{
tab[0][ken][gken]="";
}

    table[1+gken][ken+1]= tab[0][ken][gken];
    if(gken==0){
```

```
   table[1+gken][0]="n"+Integer.toString(2*dat);
   }
   else {table[1+gken][0]=Integer.toString(2*dat);
   }
    table[1+gken][0]  = table[1+gken][0].concat("");
    ken++;

    }
dat*=2;
gken++;
}

int  Lken =0;
while(Lken<i){
if(Lken==0){
table[0][1+Lken]="p"+Integer.toString((int)Math.pow(2,Lken));


}
else{
table[0][1+Lken]=Integer.toString((int)Math.pow(2,Lken));
}

table[0][1+Lken]= table[0][1+Lken].concat("");
Lken++;
}




int iiii=0;
double wari=0;

int jou2=0;
while(i>iiii)
{
int gg = 0;
while(gi>gg){
double jou=1;
wari = YY[iiii][gg];
while(1<wari){
wari = wari/10;
jou = jou *10;

}
```

```
if(jou>jou2)
{
jou2=(int)jou;

}

gg++;
}
iiii++;
}


int rr =0;
int rrr =1;
int wari2 =3;

//wari2 初期値 3
while(rr<i){
int r =0;
while(r<gi){
if((int)YY[rrr][r]!=0){
System.out.println((int)YY[rr][r]+"  "+wari2+"  " + ((int)YY[rr][r] -(int)YY[rrr][r]) + "   " +    ((int)Y
}
r++;


}
wari2 += 1;
rr++;
rrr++;
System.out.println("   ");
}


//if(gl){
Graph gp= new Graph(title, table, jou2, 0,jou2/10);
//}
}




}
```

TestpL.java

```
public class TestpL {
public static void main(String args[])
{

double n= Math.pow(2,7) ;   // データ数
int p=1;   // プロセッサ数
int L =10;   // 同期時間
int g=1;  //通信時間
 int baiL =1;
int Lk=L;
 int baiLL = L+7*baiL;

double y= 0;
 double yyy=0;



 int i=0;

int L2 =1;

double [][] YY = new double [1000][100];

double [][] YY2 = new double [1000][100];
String [][][] tab = new String [5][1000][100];
while(L < baiLL ){
int p2=p;


double count =1;
double logd=0 ;
double ap =0;
double cp = 0;
double logp =0;
int ii=0;
while(n/p2 > 1 ){
double d=n/p2;


  double lo=0;
double dd=d;
while(dd!=1){
```

```
lo++;
logd = lo;
dd/=2;

}


double lop=0;

int pp=p2;

while(pp!=1){




lop++;
logp = lop;
pp/=2;

}
     if(ii>0){
 ap = 1+logp;
}
else if(ii==0){
ap =0;
}
double bp = cp + ap;

 cp = bp;

double v =(d * logd * (bp+1));
    double x = (d*bp*g);
    double w =(bp*L);
 y =v+x+w;

YY[ii][L2-1] = y;
YY2[ii][L2-1] = v;




tab[0][ii][L2-1]  = Integer.toString((int)YY[ii][L2-1]);
tab[1][ii][L2-1] = Integer.toString(p2);
```

```
System.out.println("プロセッサ台数 " + p2 +" 同期時間 " + w + " 通信時間 " + x + " 総合計算時間 " + y);


if(ii==0){
yyy=y;

}
 else if(ii>0){
if(yyy>y){
yyy= y;
count  = Math.pow(2, ii);

}

}



 if(n==1){System.out.println(" 最速計算時間 "+yyy);
  System.out.println( " 最速の台数 " + count);
 }
p2 = p2*2;
 ii++;


}

L+= baiL;
L2++;
i= ii;
}

String [][] table = new String [L2][i+1];
String title = "プロセッサと同期";

int ten =0;
int ten2 =0;


while(ten2<L2+1){
while(ten<i+1){


table[ten2][ten]="";
ten++;
}
```

```
ten2++;
}
int gken=0;
while(gken<L2-1){
int ken =0;
while(ken<i){


    table[1+gken][ken+1]= tab[0][ken][gken];
    if(1+gken==1){

    table[1+gken][0]="L"+Integer.toString(Lk+(gken*baiL));

    }
    else{
    table[1+gken][0]=Integer.toString(Lk+(gken*baiL));
    }

    table[1+gken][0]=table[1+gken][0].concat("");
    ken++;

    }

gken++;
}


int  Lken =0;
while(Lken<i){
table[0][1+Lken]="p"+Integer.toString((int)Math.pow(2,Lken));
table[0][1+Lken]=table[0][1+Lken].concat("");
Lken++;
}




int iiii=0;
double wari=0;

int jou2=0;
while(i>iiii)
{
int gg = 0;
while(L2>gg){
int jou=1;
```

```
wari = YY[iiii][gg];
while(1<wari){
wari = wari/10;
jou = jou *10;

}

if(jou>jou2)
{
jou2=jou;
}

gg++;
}
iiii++;
}



Graph gp= new Graph(title, table, jou2, 0,jou2/10);
}









}
```

Testpg.java

```
import java.io.*;

public class Testpg {
public static void main(String args[])
{

double n= Math.pow(2,12) ;   // データ数
int p=1;   // プロセッサ数
int L=1;   // 同期時間
int g=1;  //通信時間

int baig= 100;

int baigg = 10*baig;

double y= 0;
 double yy = 0;
 double yyy=0;

int i=0;
int gi=1;


double [][] YY = new double [1000][100];
String [][][] tab = new String [5][1000][100];
while(g < baigg ){
int p2=p;


double count =1;
double logd=0 ;
double ap =0;
double cp = 0;
double logp =0;
int ii=0;
while(n/p2 > 1 ){
double d=n/p2;


  double lo=0;
double dd=d;
while(dd!=1){
```

```
lo++;
logd = lo;
dd/=2;

}


double lop=0;

int pp=p2;

while(pp!=1){




lop++;
logp = lop;
pp/=2;

}
     if(ii>0){
 ap = 1+logp;
}
else if(ii==0){
ap =0;
}
double bp = cp + ap;

 cp = bp;

double logdd = Math.log(d);
double v =(d * logd * (bp+1));
double x = (d*bp*g);
double xx = (d*bp);

double w =(bp*L);
double z = (bp * d);
double a = v+z;
 y =v+x+w;
double y2=v+w;
YY[ii][gi-1] = y;

tab[0][ii][gi-1]  = Double.toString(YY[ii][gi-1]);
```

```
tab[1][ii][gi-1] = Integer.toString(p2);

System.out.println(g+" "+v+" "+logd+" "+bp+" "+"プロセッサ台数 " + p2 +" 同期時間 " + w + " 通信時間 " + x + "
合計算時間 " + y);

yyy=y;



//   Graph gp= new Graph(タイトル，データ，ｙ軸目盛り最大値，
//                        ｙ軸目盛り最小値，ｙ軸目盛線の幅);
// 以下のように書けば，表が表示される
//   Table tb= new Table(title, table, 100, 0, 10);

if(ii==0){
yyy=y;

}
 else if(ii>0){
if(yyy>y){
yyy= y;
count  = Math.pow(2, ii);

}

}



 if(n==1){System.out.println(" 最速計算時間 "+yyy);
  System.out.println( " 最速の台数 " + count);
 }
p2 = p2*2;
 ii++;


}

g+= baig;
gi++;
i= ii;
}

String [][] table = new String [gi][i+1];
String title = "プロセッサ";
String XX="";
```

```
int ben =1;
int ten =0;
int ten2 =0;


while(ten2<gi){
while(ten<i+1){


table[ten2][ten]="";
ten++;
}
ten2++;
}
int gken=0;

while(gken<gi-1){
int ken =0;
while(ken<i){


    table[1+gken][ken+1]= tab[0][ken][gken];
    table[1+gken][0]=Integer.toString(1+(gken*baig));
    table[1+gken][0]= table[1+gken][0].concat(" g");
    ken++;

    }
gken++;
}

int  Lken =0;
while(Lken<i){
table[0][1+Lken]=Double.toString(Math.pow(2,Lken));
table[0][1+Lken] =table[0][1+Lken].concat(" p");
Lken++;
}




int iiii=0;
double wari=0;

int jou2=0;
while(i>iiii)
{
```

```
int gg = 0;
while(gi>gg){
int jou=1;
wari = YY[iiii][gg];
while(1<wari){
wari = wari/10;
jou = jou *10;

}

if(jou>jou2)
{
jou2=jou;
}

gg++;
}
iiii++;
}



Graph gp= new Graph(title, table, jou2, 0,jou2/10);
}









}
```