卒業研究報告書

題 目

BSPモデルを用いた
最小スパニング木

指 導 教 員

石 水　隆 助手

報 告 者

02–1–47–134

小 林　洋 亮

近畿大学理工学部情報学科

平成 18 年 2 月 10 日提出

概要

本研究では代表的なグラフ問題である最小スパニング木 (MST:Minimum Spanning Tree) 問題を分散メモリ型並列計算モデルである BSP(Bulk Synchronous Parallel) モデル上で解く並列アルゴリズムを提案する。

最小スパニング木問題はガス管の配管や通信回線の架設などに応用できる問題である。よって並列計算機上で実行できるアルゴリズムを開発することは重要である。また BSP モデルとは分散メモリ型の並列計算モデルであり、従来の PRAM（Parallel Random Access Machine）と異なり、最近の並列計算において重要視されている通信遅延及び同期時間を考慮したモデルである。PRAM アルゴリズムでは通信及び同期が考慮されていないため、これをそのまま BSP モデル上で実行させた場合、効率良く実行できるとは限らない。従って、BSP モデル用の通信及び同期を考慮した並列アルゴリズムが必要となる。

本研究では頂点数 n の重み付き無向グラフに対し BSP モデル上で n プロセッサを用いて $O(nglogn + Llog^2n)$ 時間で最小スパニング木問題を解く並列アルゴリズムを提案する。ここで g は 1 メッセージあたりの通信時間、L は同期時間である。

目 次

1 序論 **1**
1.1 本研究の背景 1
1.1.1 並列処理 1
1.1.2 並列アルゴリズム 1
1.1.3 並列処理の必要性 1
1.1.4 並列計算モデル 2
1.1.5 最小スパニング木 2
1.2 本研究の目的 3
1.3 本報告書の構成 3

2 準備 **4**
2.1 PRAM(Parallel Random Access Machie) モデル 4
2.2 BSP(Bulk-Synchronous Parallel) モデル 4
2.3 最小スパニング木 (Minimum Spaning Tree:MST) 問題 7
2.4 グラフ表現 7
2.5 2 分木探索を用いた最小値演算 8

3 BSP モデル上での最小スパニング木 (MST) アルゴリズム **9**
3.1 アルゴリズム 9
3.2 計算量 14

4 結果・考察 **17**

5 結論・今後の課題 **18**

6 謝辞 **19**

1　序論

1.1　本研究の背景

1.1.1　並列処理

我々は、「一人で処理するには時間のかかる仕事を，複数人で協力することによって短時間で処理する」ということをよく行う。これと同じように,「一台のプロセッサで実行するには時間のかかる処理を、複数のプロセッサで協力することによって短時間で実行する」という処理が並列処理 (Parallel Processing) であり、それを行うためのアルゴリズムが並列アルゴリズム (Parallel Algorithm) である。

1.1.2　並列アルゴリズム

アルゴリズムとは、ある目的を実現するために必要な作業の手順を、明確に述べたもののことを指す。この手順とは、適当にやるというのではなく、その指示に正確に従えば誰でも全く同じ結果が得られるように詳しく述べられていなければならない。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、本論文は計算機に問題を解かせるための手順をさす。

並列アルゴリズム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、一般的な逐次アルゴリズムとは別の並列計算モデル用の新たなアルゴリズムが必要である。

1.1.3　並列処理の必要性

今日、多くの分野においてコンピュータによる高速処理や大規模データ処理が必要とされている。計算に要する時間を短縮するためには、速いコンピュータが必要となるが、現在のハードウェアを作る技術（半導体素子の作成技術）は限界に近づきつつあり、これ以後何千倍も高速化されることはないと考えられている。そこで、この問題の解決方法としてソフトウェアの点で注目されているのが並列処理の技術である。また近年コンピュータ素子の価格とサイズが急激に下がっており、多数のプロセッサを用いた並列コンピュータがつくりやすくなっている。このような理由により、現在並列処理の必要性が高まってきている。

1.1.4 並列計算モデル

並列アルゴリズムを考えるとき、並列アルゴリズムを実行するための計算機環境をある程度簡単に捉えた並列計算モデル (Parallel Computing Model) が必要になる。並列計算モデルには PRAM (Parallel Random Access Machine) モデル [2]、BSP (Bulk Synchronous Parallel) モデル [4] など、様々なモデルがあり、そしてそれぞれのモデルに対して効率的な並列アルゴリズムが考えられている。

初期の並列計算モデルは並列計算機 1 台のプロセッサの演算能力が低かったため、プロセッサ内部の演算に比べて、プロセッサ間の通信はそれほど考慮が必要とされておらず、通信コストの表現には重点がおかれておらず、そのモデルとして PRAM が用いられた。しかし近年では、プロセッサの演算能力の向上とプロセッサ数の大規模化により、並列処理における通信時間は無視できない要素となってきた。

また近年は複数の計算機をネットワーク接続し、それ全体を 1 台の仮想的な並列計算機として用いるクラスタ（Cluster) 処理やグリッド (Grid) 処理もさかんに行われており、これらの処理では通信時間および同期時間が重大な要素となる。本論文では最近の並列計算において重要視されている通信遅延及び同期時間を考慮する事の出来る BSP モデルを扱う。

BSP モデルとは Valiant により提案された分散メモリ型の並列計算モデルであり、従来の PRAM では考慮されていなかった、通信および同期に掛かるコストを 1 メッセージ辺りの送受信時間 g, 同期時間 L というパラメタにより表すことを可能としたモデルである。

1.1.5 最小スパニング木

各辺に正の重みが付加された重み付き連結無向グラフ G に対して G の全ての頂点を含む G の部分木を G のスパニング木 (Spaning Tree) と言い、スパニング木のうち辺の重みの和が最小のものを G の最小スパニング木 (Minimum Spaning Tree) という。

最小スパニング木問題はガス管の配管や通信回線の架設などに応用できる問題である。よって並列計算機上で実行できるアルゴリズムを開発することは重要である。

頂点数 n のグラフに対し、Prim は $O(n^2)$ で最小スパニング木を求めるアルゴリズムを提案した。また Sollin は CRCW-PRAM 上で $O(log^2)$ 時間 n^2 プロセッサの並列アルゴリズムを提案した。

1.2 本研究の目的

本研究では代表的なグラフ問題である最小スパニング木 (MST:Minimum Spanning Tree) 問題を BSP モデル上で解く並列アルゴリズムを提案する。

1.3 本報告書の構成

2 節では本研究を始める準備として、必要な知識である PRAM モデル、BSP モデル、最小スパニング木問題などの詳しい説明を記述する。また 3 節では本研究で提案した BSP モデル上でのアルゴリズムを記述し、その計算量についても記述する。4 節では 3 節で提案したアルゴリズム計算評価、及び BSP モデルについての考察を記述する。5 節では本研究の結論について記述し、そこから今後考えられる課題について記述する。

2 準備

2.1 PRAM(Parallel Random Access Machie) モデル

PRAM(Parallel Random Access Machie)[2] は共有メモリ型並列計算機を抽象化した並列計算モデルである。PRAM の概念図を図 1 に示す。共有メモリ型並列計算機では，プロセッサ間でのデータのやりとりは共有メモリへの読み書きで行われる。PRAM モデルは，共有メモリへのアクセスの際，プロセッサ間でアクセスの競合が一切起こらないという、極めて理想的な完全同期型モデルである。そのため，このモデルではプログラムに存在する全ての並列性の抽出が可能となっている。しかし，共有メモリ型並列計算機を対象としているために分散メモリ型並列計算機には適していない。また，プロセッサ間の通信のオーバヘッドを考慮してなかったりや同期オーバヘッドや記憶へのアクセス競合が無視されている点など実際の並列計算機と異なる点が多く，実行時間予測に用いるには実用的ではない。

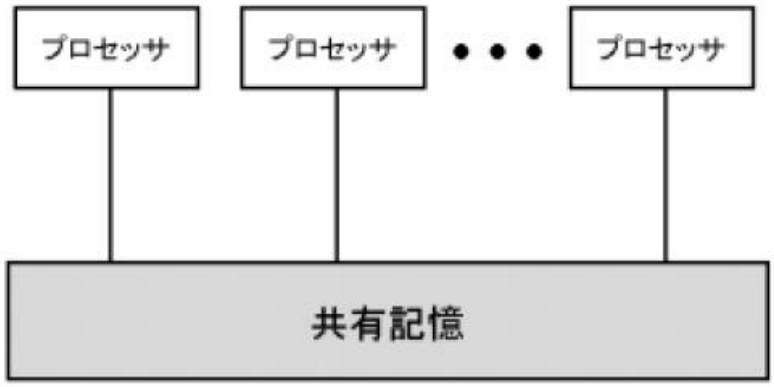

図 1: PRAM(Parallel Random Access Machie)

2.2 BSP(Bulk-Synchronous Parallel) モデル

BSP(Bulk-Synchronous Parallel) モデル [4] 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

BSP モデルは、図 2 のように、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。計算と通信の間にバリア同期があり、計算と通信が交互に行われるモデルである。PRAM モデルと比べると、同期や通信が考慮に入れられていて、より実際の並列計算機の近いモデルになっている。

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:バリア同期時間および通信遅延時間
- $g(\leq L)$: 1 個の送信命令または受信命令の実行に必要な時間
- p:プロセッサ数 各プロセッサは　0〜$p-1$ のプロセッサ番号を持ち、$p_i(0 \leq i < p)$ と表わされる。

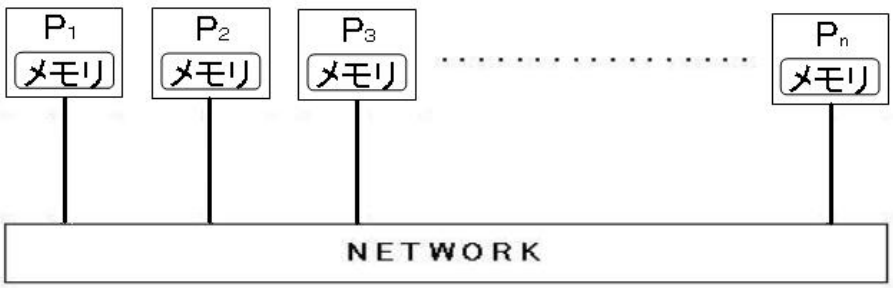

図 2: BSP(Bulk-Synchronous Parallel) モデル

BSP モデル上で各プロセッサが処理する様子を図 3 に示す

各プロセッサが実行する処理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が高々w 個の内部計算命令と、高々h 個の通信命令を実行するとき、そのスーパーステップの時間計算量は $O(w+$

$gh + L$) 時間と仮定されている。BSP アルゴリズムの時間計算量は、各スーパーステップの時間計算量の和として表わされる。

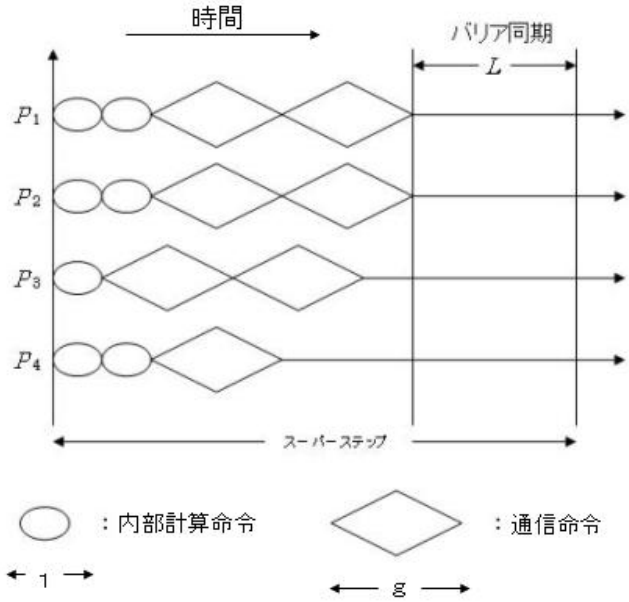

図 3: BSP モデルにおけるスーパーステップの例

2.3 最小スパニング木 (Minimum Spaning Tree:MST) 問題

各辺に正の重みが付加された連結無向グラフ G に対して、G の全ての頂点を含む G の部分木を G のスパニング木 (Spaning Tree) と言い、辺の重みの和が最小となるスパニング木を最小スパニング木 (Minimum Spaning Tree) と言う。最小スパニング木問題とは、重み付連結無向グラフ G が与えられたとき、その最小スパニング木を求める問題である。図 4 に示すグラフ G が与えられたとする。このとき図 4 の太線部分の辺が最小スパニング木の辺である。

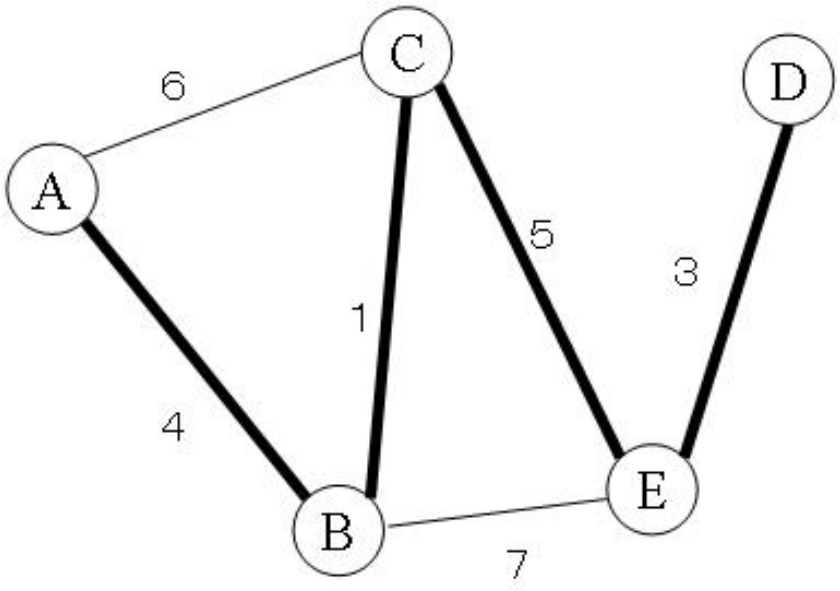

図 4: 重み付きグラフとその最小スパニング木の例

Prim[3] は $O(n^2)$ 時間で最小スパニング木問題を解く逐次アルゴリズムを提案した。また Sollin[1] は CREW − PRAM 上で $O(log^2 n)$ 時間、$O(n^2)$ プロセッサの並列アルゴリズムを提案した。

2.4 グラフ表現

本研究ではグラフのデータの表現方法として、隣接行列を用いる。

隣接行列とはグラフ G に対して頂点 i と頂点 j の間の重みを表現するのに $a[i][j]$ に重みを格納した 2 次元配列 a となる。

図 5 の無向グラフを隣接行列を使用して表現すると表 1 となる。

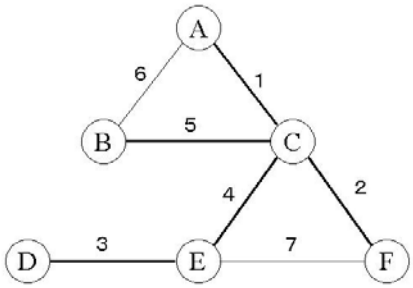

図 5: グラフ G

	A	B	C	D	E	F
A	∞	6	1	∞	∞	∞
B	6	∞	5	∞	∞	∞
C	1	5	∞	∞	4	2
D	∞	∞	∞	∞	3	∞
E	∞	∞	4	3	∞	7
F	∞	∞	2	∞	7	∞

表 1: 隣接行列

2.5　2 分木探索を用いた最小値演算

n 個のデータに対して、2 分木探索を用いて最小値演算する方法を以下に示す。

1. データを 2 個ずつの組にする。
2. 各組において値の小さいものを選択する。
3. １．２．の動作を $logn$ 回繰り返す。以上の操作により n 個のデータから最小値を求めれる。

この 2 分木探索は並列処理において、1 台のプロセッサに負荷がかからないように、プロセッサにかかる負荷を分散させる役割がある。

3 BSP モデル上での最小スパニング木 (MST) アルゴリズム

3.1 アルゴリズム

本研究で提案する BSP モデル上で MST 問題を解く並列アルゴリズムは Sollon のアルゴルズム [1] をベースとしている。

以下に本研究で提案した BSP モデル上で MST 問題を解くアルゴリズム **BSP-MST** およびアルゴリズム BSP-MST 中で使用する手続き **find_parent**, **pointer_jump**, **data_collect**,**remake_graph** を示す。

(アルゴリズム **BSP-MST**)

入力: 頂点数 n の重み付無向グラフ $G = (V, E)$。プロセッサ P_i $(0 \leq i < n)$ が頂点 i および辺 (i, j) $(0 \leq j < n)$ から成る G の部分グラフ G_i を保持する。(G は $n \times n$ の隣接行列 A として表され、プロセッサ P_i $(0 \leq i < n)$ は A の i 行目から成る部分配列 A_i を保持する。)

出力: G の最小スパニング木 $T = (V, E)$。プロセッサ P_i $(0 \leq i < n)$ が頂点 i から出る T の辺 (i, j) $(0 \leq j < n, (i, j) \in T)$ から成る T の部分グラフ T_i を保持する。(T は $n \times n$ の隣接行列 B として表され、プロセッサ P_i $(0 \leq i < n)$ は B の i 行目から成る部分配列 B_i を保持する。)

1. $p = n$ とし、プロセッサ $P_0, P_1, ..., P_{p-1}$ を用いて、以下の操作を $p = 1$ になるまで繰り返す。
   (a) プロセッサ P_i $(0 \leq i < p)$ は手続き find_parents を用いて頂点 i の親 parent(i) を決定する。
   (b) プロセッサ P_i $(0 \leq i < p)$ は辺 $(i, \text{parent}(i))$ を T_i に加える
   (c) プロセッサ P_i $(0 \leq i < p)$ は手続き pointer_jump を用いて頂点 i の根 root(i) を決定する。
   (d) プロセッサ P_i $(0 \leq i < p)$ は、プロセッサ $P_0, P_1, ..., P_{p-1}$ に頂点 i の根の頂点番号 root(i) をブロードキャストする。
   (e) 頂点 i が根ではないプロセッサ P_i $(0 \leq i < p, i \neq \text{parent}(i))$ は手続き collect_data を用いて頂点 i から出る G の辺 (i, j) $(0 \leq j < p \wedge (i, j) \in G_i)$ および頂点 i から出る T の辺 (i, j) $(0 \leq j < p \wedge (i, j) \in T_i)$ の情報をプロセッサ $P_{\text{root}(i)}$ に集約する。
   (f) プロセッサ P_i $(0 \leq i < p)$ は手続き remake_graph を用いて同一の根を持つ頂点集合を 1 つの頂点とする新たなグラフを作る。 □

(手続き **find_parent**)

p 台のプロセッサを用いて以下の処理を行う。

1. プロセッサ P_i $(0 \leq i < p)$ は、頂点 i から出る辺 (i, j) $(0 \leq j < p, (i, j) \in G$ のうち最も重みが軽い辺 (i, k) を選ぶ。このとき、頂点 i の親を頂点 k とし、$\mathrm{parent}(i) = k$ とする。

2. プロセッサ P_i $(0 \leq i < p)$ は、プロセッサ $P_{\mathrm{parent}(i)}$ に頂点番号 i を送信する。

3. プロセッサ P_i $(0 \leq i < p)$ は、2. でプロセッサ番号を送信してきたプロセッサに対し、頂点 i の親 $\mathrm{parent}(i)$ を送信する。

4. プロセッサ P_i $(0 \leq i < p)$ は、頂点 i の親の親が i 自身 $(i = \mathrm{parent}(\mathrm{parent}(i)))$ であり、かつ頂点 i の番号が頂点 i の番号よりも若い $(i < \mathrm{parent}(i))$ 場合、頂点 i の親を頂点 i 自身に変更し、$\mathrm{parent}(i) = i$ とする。 □

(手続き **pointer_jump**)
p 台のプロセッサを用いて以下の処理を行う。

1. 以下の処理を $\log p$ 回繰り返す

   (a) プロセッサ P_i $(0 \leq i < p)$ は、プロセッサ $P_{\mathrm{parent}(i)}$ に頂点番号 i を送信する。

   (b) プロセッサ P_i $(0 \leq i < p)$ は、1.(a) でプロセッサ番号を送信してきたプロセッサに対し、頂点 i の親 $\mathrm{parent}(i)$ を送信する。

   (c) プロセッサ P_i $(0 \leq i < p)$ は、頂点 i の親を頂点 i の親の親に変更し、$\mathrm{parent}(i) = \mathrm{parent}(\mathrm{parent}(i))$ とする。

2. プロセッサ P_i $(0 \leq i < p)$ は、頂点 i の根を頂点 i の親とし、$\mathrm{root}(i) = \mathrm{parent}(i)$ とする。 □

1.(a) から 1.(c) が繰り返される例を図 6 に示す。

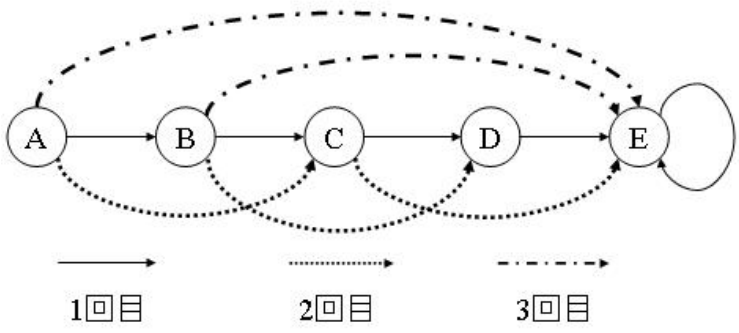

図 6: ポインタジャンプ過程

(手続き **collect_data**)
p 台のプロセッサを用いて以下の処理を行う。

1. プロセッサ P_i $(0 \leq i < p)$ は、プロセッサ $P_{\mathrm{root}(i)}$ に頂点番号 i を送信する

2. プロセッサ P_i $(0 \leq i < p)$ が 1. で頂点番号を受信した場合、受信した頂点番号および頂点 i 自身から成る集合を C_i とする。(プロセッサ P_i は、頂点 i の根が頂点 i 自身である場合 $(i = \mathrm{root}(i))$ のみ 1. で頂点番号を受信する。)

3. 以下の操作を C_i に含まれる番号が 1 個になるまで繰り返す。

   (a) 頂点 i が根であるプロセッサ P_i $(0 \leq i < p \wedge i = \mathrm{roor}(i))$ は、頂点番号集合 C_i の中の頂点を 2 つずつ組にする。頂点 $j \in C_i$ と頂点 $k \in C_i$ が組である場合、頂点 k を頂点 j のパートナーと呼び、$\mathrm{partner}(j) = k$ とする。

   (b) 頂点 i が根であるプロセッサ P_i $(0 \leq i < p)$ は、プロセッサ P_j $(0 \leq j < p \wedge j \in C_i)$ に対し、頂点番号 $\mathrm{partner}(j)$ を送信する。

   (c) プロセッサ P_i $(0 \leq i < n)$ は、頂点 i から出る G の辺 (i, j) $(0 \leq j < n \wedge (i, j) \in G_i)$ および頂点 i から出る T の辺 (i, j) $(0 \leq j < n \wedge (i, j) \in T_i)$ の情報をプロセッサ $P_{\mathrm{partner}}(i)$ に送信する。

   (d) プロセッサ P_i $(0 \leq i < n)$ は、頂点 i から出る G の辺 (i, j) および頂点 $\mathrm{partner}(i)$ から出る G の辺 $(\mathrm{partner}(i), j)$ $(0 \leq j < n \wedge (i, j) \in G_i \wedge (\mathrm{partner}(i), j) \in G_{\mathrm{partner}(i)})$ に対し、辺 $(\mathrm{partner}(i), j)$ の重みが辺 (i, j) の重みよりも小さいならば辺 (i, j) の重みを辺 $(\mathrm{partner}(i), j)$ の重みに置き換える。

   (e) 頂点 i が根であるプロセッサ P_i $(0 \leq i < n \wedge i = \mathrm{roor}(i))$ は、C_i の中の各頂点組 (j, k) $(j, k \in C_i)$ に対して、頂点 j または頂点 k のどちらかを頂点番号集合 C_i より削除する。ただし、頂点 j または k が頂点 i 自身であるときは、頂点 i のパートナーを頂点番号集合 C_i より削除する。

4. 頂点 j_k $(k \geq 1)$ を $\mathrm{root}(j_k) = j$ である頂点とする。頂点 i が根であるプロセッサ P_i $(0 \leq i < p \wedge i = \mathrm{roor}(i))$ は、頂点 i から根である頂点 j $(0 \leq j < p \wedge j = \mathrm{roor}(j))$ への辺 (i, j) に対し、辺 (i, j) の重みを辺 $(i, j_1), (i, j_2), \ldots$ うち最も重みの小さい辺の重みに置き換える。

5. 頂点 i が根であるプロセッサ P_i $(0 \leq i < p \wedge i = \mathrm{roor}(i))$ は、頂点 i から根ではない頂点 j $(0 \leq j < p \wedge j \neq \mathrm{roor}(j))$ へ向かう辺 (i, j) を G_i から削除する。 □

(手続き **remake_graph**)
p 台のプロセッサを用いて以下の処理を行う。

1. プロセッサ P_i $(0 \leq i < p)$ は、頂点 i が根であるならば $d_i = 1$、そうでなければ $d_i = 0$ とする。
2. 全てのプロセッサで $d_0, d_1, ..., d_{p-1}$ の接頭部和を求める。$d_0, d_1, ..., d_{p-1}$ の接頭部和を $e_0, e_1, ..., e_{p-1}$ とする。$(e_i = \sum_{k=0}^{i} d_k)$
3. プロセッサ P_{p-1} は $e_{p-1} = \sum_{k=0}^{p-1} d_k$ をプロセッサ $P_0, P_1, ..., P_{p-1}$ にブロードキャストする。
4. 根である頂点 i を持つプロセッサ P_i $(0 \leq i < p \wedge i = \mathrm{root}(i)$ は、頂点 i から出る G の辺 (i, j) $(0 \leq j < n \wedge (i, j) \in G_i)$ および頂点 i から出る T の辺 (i, j) $(0 \leq j < n \wedge (i, j) \in T_i)$ の情報をプロセッサ $P_{e_i - 1}$ に送信する。
5. $p = e_{p-1}$ とする。

□

例 1 (アルゴリズムの実行例)

図 5 に示すグラフ G が与えられたとする。手続き find_parent を用いることにより、各頂点の親が決定される。図 6 は手続き find_parent の 1. 終了時点での各頂点が持つ親へのポインタである。手続き pointer_jump を用いることにより各頂点の根が決定される。図 7 は手続き pointer_jump 終了後の各頂点が持つ根へのポインタである。手続き data_collect を用いることにより各頂点が持つ辺のデータは根に集約され、は手続き remake_graph を用いることにより同じ根を持つ頂点集合全体を 1 つの頂点とする図 8 のような新たなグラフが作られる。以上の操作を頂点 1 つになるまで繰り返し、図 9 の最小スパニング木が生成できる。

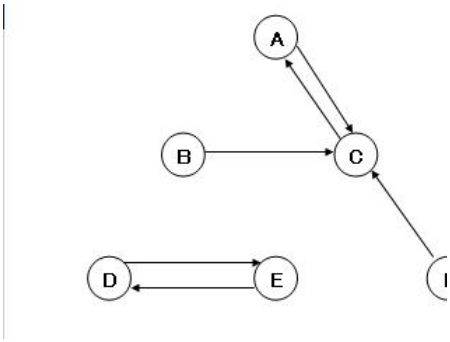

図 7: 最小の頂点の選択

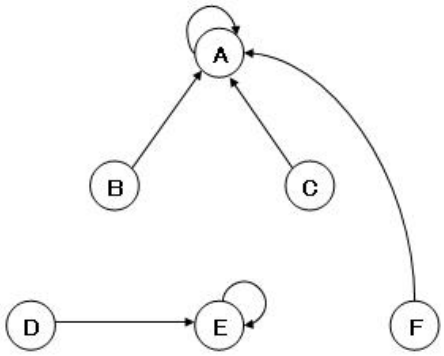

図 8: ポインタジャンプによる根付き有向スターの作成

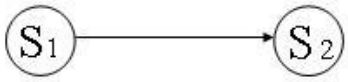

図 9: データコレクトによりそれぞれのスターを最小の重みで接続

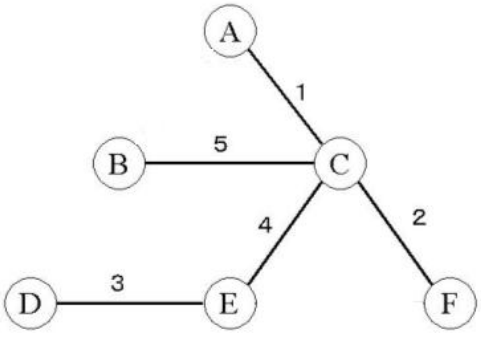

図 10: 最小スパニング木

3.2 計算量

本研究で提案したアルゴリズム BSP-MST 中の各手続きについて以下の補題が成り立つ。

補題 1 手続き find_parent は BSP モデル上で p プロセッサを用いて $O(gp+L)$ 時間で各頂点の親を決定できる。
(証明)
手続き find_parent の各ステップの計算量を検証する。
1. はプロセッサ内部が持つデータに対する最小値演算であるので $O(p)$ 時間で実行できる。
2. は各プロセッサで 1 個の頂点番号を送信し、高々p 個の子の頂点番号を受信する。また 3. は各プロセッサで高々p 個の親の頂点番号を送信し、1 個の親の親の頂点番号を受信する。よって 2. と 3. は $O(gp+L)$ 時間で実行できる。
4. は各プロセッサで高々定数個の内部計算を行うので、$O(1)$ 時間で実行できる。
よって find_parent は $O(gp+L)$ 時間で実行できる。 □

補題 2 手続き pointer_jump は BSP モデル上で p プロセッサを用いて $O((gp+L)logp)$ の時間で各頂点を根につなげることが出来る。
(証明)
手続き pointer_jump の各ステップの計算量を検証する。
1.(a) は各プロセッサで 1 個の頂点番号を送信し、高々p 個の子の頂点番号を受信する。また 1.(b) は各プロセッサで 1 個の親の頂点番号を送信し、1 個の親の親の頂点番号を受信する。よって 1.(a) と 1.(b) は $O(gp+L)$ 時間で実行できる。
1.(c) は高々定数個の内部計算を行うので、$O(1)$ 時間で実行できる。よってこの 1. の作業を $logp$ 回繰り返すので、$O((gp+L)logp)$ の時間で実行できる。
2. も 1.(c) と同様に高々定数個の内部計算を行うので、$O(1)$ 時間で実行できる。
よって pointer_jump は $O((gp+L)logp)$ の時間で実行出来る。 □

補題 3 手続き collect_data は BSP モデル上で p プロセッサを用いて $O(gp+Llogp)$ の時間で各頂点を根につなげることが出来る。
(証明)
手続き collect_data の各ステップの計算量を検証する。1. は各プロセッサで 1 個の頂点を送信し、根であるプロセッサは高々p 個のデータを受信するので、$O(gp+L)$ の時間で実行出来る。
2. は根を持っているプロセッサが高々p 個のデータを集合として格納していくので、内部計算 $O(p)$ 時間で実行出来る。

3.(a) は根の情報を持っているプロセッサによる内部計算で、適当に 2 つの組み合わせを作るので、$O(p)$ 時間で実行出来る。

3.(b) は子であるプロセッサで 1 個の頂点番号を受信し、根であるプロセッサは高々p 個の頂点に 1 個のデータを送信するので、$O(gp+L)$ の時間で実行出来る。

3.(c) は各プロセッサが高々p 個のデータを partner である 1 個のプロセッサに情報を送るので、$O(gp+L)$ の時間で実行出来る。

3.(d) は各プロセッサによる大小比較の高々定数個の内部計算を行うので、$O(1)$ の時間で実行出来る。

3.(e) は高々定数個の内部計算なので、$O(1)$ の時間で実行出来る。

3.(a)〜3.(e) を頂点が根だけになるまで $logp$ 回繰り返す。よって $O((gp+L)logp)$ の時間が必要だと考えられるが、この操作は 1 回行うことによって頂点の数が $\frac{1}{2}$ になるよって $(gp+L)+(\frac{1}{2}gp+L)+(\frac{1}{4}gp+L)+\cdots+(g+L) = O(gp+Llogp)$ の時間で実行出来る。

4. は各根のプロセッサは内部の持つ最小値演算を行うので $O(p)$ の時間で実行出来る。

5. は各根のプロセッサは内部計算を高々$O(p)$ 回繰り返すので、$O(p)$ の時間で実行出来る。

よって collect_data は $O(gp+Llogp)$ の時間で実行出来る。 □

補題 4 手続き remake_graph は BSP モデル上で p プロセッサを用いて $O(gp+L)$ 時間で各頂点の親を決定できる。

(証明)

手続き remake_graph の各ステップの計算量を検証する。

1. は各プロセッサで高々定数個の内部計算を行うので、$O(1)$ 時間で実行できる。

2. は各プロセッサで 1 個のデータを送信し、高々p 個の子のデータを受信する。よって通信に必要な時間は $O(gp+L)$ 内部計算必要な時間は $O(p)$ である。よって 2. は $O(gp+L)$ の時間で実行出来る。

3. 各プロセッサは 1 個のデータを高々p 個のプロセッサに送信する。よって $O(gp+L)$ の時間で実行出来る。

4. 各プロセッサは高々p 個のデータを任意のプロセッサに送信する。よって $O(gp+L)$ の時間で実行出来る。

5. は各プロセッサで高々定数個の内部計算を行うので、$O(1)$ 時間で実行できる。

よって remake_graph は $O(gp+L)$ の時間で実行出来る。 □

補題 1〜4 より、以下の定理が成立する。

定理 1 アルゴリズム *BSP-MST* は、*BSP* モデル上で n プロセッサを用いて $(nglogn+Llog^2n)$ 時間で *MST* 問題を解く。

(証明)

各繰り返しにおいて、繰り返し終了時の頂点数は繰り返し開始時の頂点の高々$\frac{1}{2}$ である。よって、i 回目の繰り返しの開始時において頂点数は高々$\frac{n}{2^{i-1}}$ となる。よって、i 回目の繰り返しの時間計算量は $O((g\frac{n}{2^{i-1}} + L)logn)$ となり、アルゴリズム全体の時間計算量は $(nglogn + Llog^2n)$ となる。

□

4　結果・考察

表 2 に頂点数 n の重み付連結無向グラフの最小スパニング木問題を解く逐次計算のアルゴリズムである Prim のアルゴリズム [3]、PRAM モデル [2] 上のアルゴリズムである Sollin[1] のアルゴリズムおよび本研究で提案した BSP モデル用に改良したアルゴリズムの計算量を示す。

Sollin のアルゴリズムが CREW − PRAM 上でプロセッサ数 n^2 台であるのに対し、本研究で提案したアルゴリズムはプロセッサ数 n 台である。

BSP モデル上で大きいプロセッサを使用すると、各プロセッサはデータを少しづつ持ち、通信命令、バリア同期が増加してしまい、効率の良いアルゴリズムを提案するのが、困難になることがある。よってプロセッサを増やすことにより、計算時間も増加してしまう可能性も考えられる。また BSP モデルのアルゴリズムを効率良くする方法として、１つのプロセッサにデータが集まり過ぎると、そのプロセッサの受信命令に時間がかかりすぎてしまうので、データの負荷分散を行うことにより通信時間を減らせることが分かる。

モデル	計算量	プロセッサ	文献
RAM	$O(n^2)$	1	[3]
PRAM	$O(log^2 n)$	n^2	[1]
BSP	$O(nglogn + Llog^2 n)$	n	本研究

表 2: MST を解くアルゴリズムの計算量

5　結論・今後の課題

本研究では、重み付連結無向グラフ G が与えられたとき、最小スパニング木問題を BSP モデル上で解く並列アルゴリズムを提案した。

本研究で提案したアルゴリズムは、n 頂点の重み付連結無向グラフ G に対して、n プロセッサを用いて $O(ng \log n + L \log^2 n)$ 時間で最小スパニング木問題を解く。

より大きいプロセッサ台数を使用して通信命令、バリア同期を如何に少なくして効率の良い最小スパニング木問題を解く BSP モデル上の並列アルゴリズムの設計が今後の課題である。

6　謝辞

本研究をするにあたり、石水隆先生にはアルゴリズム、計算方法やコンピュータについて夜遅くまで御指導、御支援していただき、大変お世話になりました。誠に感謝を申し上げます。また、同研究室の皆にもお世話になり、誠に感謝申し上げます。

参考文献

[1] C.Berge and A.Ghouila-Houri: "Programming, Games and Transportation Networks," John Wiley (1965).

[2] J.JáJá：An Introduction to Parallel Algoritms (1992).

[3] R.C.Prim:"Shortest connection networks and some generalizations, " Journal of Bell System Tech, Vol.36, No.6, pp.1389–1401 (1957).

[4] L.G.Valiant："A Bridging Model for Parallel Computation," Comm. of the ACM, Vol.33, No.8, pp.103–111 (1990).