基般 解説

BSP（Bulk Synchronous Parallel）モデル再訪

―BSP によるマルチコアプログラミングと大規模グラフフレームワーク―

松崎 公紀（高知工科大学） 江本 健斗（九州工業大学）

2004 年に Google から提案された MapReduce[☆1] は，大規模データ処理フレームワークとして注目され，今ではそのオープンソース実装の Hadoop MapReduce[☆2] が広く利用されている．その後，2010 年には Google から Pregel という大規模グラフ処理のためのフレームワークが提案された．こちらも同様に Giraph[☆3] や GPS[☆4] といったオープンソース実装が作られている．MapReduce はその論文において「関数プログラミングにインスパイアされた」と述べられたが，同様に，Pregel は「Bulk Synchronous Parallel（BSP）モデルにインスパイアされた」と述べられている．これにより，BSP という計算モデルに関心が持たれるようになった．

BSP は，1990 年に L. G. Valiant[☆5] によって提案された（分散メモリ型の）並列計算機のモデルであり，またその上の並列計算モデルでもある．この抽象化された並列計算機モデルの上で，これまでさまざまなアルゴリズムが提案されている．また BSP の計算モデルをサポートするライブラリを用いると，そのような並列アルゴリズムをおよそそのままの形で実現することができる．

本稿では，もともとの BSP の提案とその前後の歴史を振り返ることから始める．その後，マルチコア環境で利用できる BSP プログラミングのライブラリの 1 つ“MulticoreBSP for C”と，BSP に基づく大規模グラフ処理フレームワークの 1 つ GPS について，具体的なコードを交えて解説する．

☆1 http://research.google.com/archive/mapreduce.html
☆2 http://hadoop.apache.org/
☆3 http://giraph.apache.org/
☆4 http://infolab.stanford.edu/gps/
☆5 L. G. Valiant は，主に機械学習や計算量に関する理論的研究の功績により，2010 年 ACM Turing 賞を受賞している．

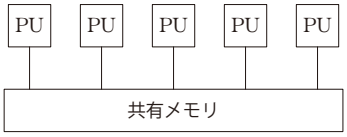

図 -1 PRAM の計算機モデル

❑ BSP モデルへ至る歴史

並列アルゴリズムに関する研究は，古くは 1970 年代より行われている．1980 年代には，主として並列ランダムアクセス計算機（PRAM）に基づく並列アルゴリズムの研究が多数行われている．PRAM は，**図 -1** に示すような共有メモリ型の計算機をモデル化したものであり，各処理装置（PU）は同期して動作する（共有メモリへのアクセスが同時に可能かどうかによっていくつかの分類がある）．PRAM に基づく並列アルゴリズムはさまざま提案されたが，PRAM の計算機モデルと実際の並列計算機とのギャップが大きいことが問題として認識されるようになった．

このような背景のもと，1990 年に L. G. Valiant によって提案されたものが BSP モデルである [1]．L. G. Valiant による論文のタイトルの“Bridging Model”という言葉にある通り，BSP モデルが当初目指したものは，より現実的な並列計算機（と並列計算）のモデルを作ることによって，ハードウェアとソフトウェアの間の橋渡しをすることである．BSP では，**図 -2** に示すようなネットワークによってつながれた分散メモリ型の計算機をモデル化して

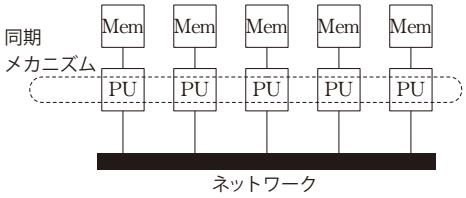

図 -2　BSP の計算機モデル

いる．

Valiant による提案の後，Valiant と B. McColl を中心に研究が進められ，1996 年には BSP プログラミングのためのライブラリの標準 BSPlib☆6 が定義されるに至った．2010 年までに，世界中で 18 の研究グループで BSP に関係する研究が行われている☆7．

1990 年前半には，BSP のほかにも，D. Culler による LogP や，F. Dehne による CGM などの類似の抽象並列計算機モデルが提案されている [5]．それらの中で，BSP が研究者の注目を最も集めたのは，BSP による並列計算機の抽象化がより使いやすかったことや，BSPlib のようなライブラリ標準が策定されたことにあると考える．

❑ BSP の計算機モデル

BSP では，抽象化された並列計算機モデルとその上の計算モデルの 2 つが与えられている．本章では，まず BSP の計算機モデルと計算モデルを紹介し，さらに BSP の特徴について述べる．

共有メモリ型の並列計算機を抽象化した PRAM と異なり，BSP は図 -2 に示すようなネットワークで接続された分散メモリ型の並列計算機を抽象化している．BSP の計算機モデルは，局所的なメモリを持つプロセッサ（PU），ネットワーク，バリア同期メカニズムの 3 つを構成要素として持つ．各プロセッサは独立に計算を行い，プロセッサ間のデータのやりとりはネットワークを通じて行われる．さらに，バリア同期メカニズムがあることが BSP の計算機モデルの特徴である．

☆6 http://www.bsp-worldwide.org/implmnts/oxtool/bsplib.html
☆7 http://www.bsp-worldwide.org/

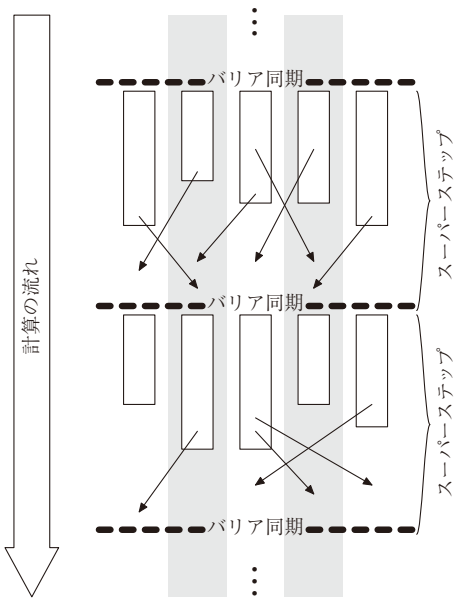

図 -3　BSP の計算モデル

計算機モデルにバリア同期があることは，BSP の計算モデルの定義にも大きな影響がある．BSP の計算モデルは，**図 -3** に示すような，バリア同期によって区切られたスーパーステップ（superstep）と呼ばれる部分の繰り返しからなる．それぞれのスーパーステップでは，各プロセッサは局所的なメモリ上にある値を用いて計算を行い，他のプロセッサへのデータ送信を行う（図では，白と灰色の帯で各プロセッサに対応する部分を表している．また，白の四角は局所的な計算，矢印はデータ通信を表す）．ここで，計算とデータ送信はどの順で行ってもよいが，あるスーパーステップにおいて他プロセッサから送られたデータを同じスーパーステップ内で計算に使うことはできない．それぞれのスーパーステップの終わりにはバリア同期があり，それによりプロセッサ間の通信の終了が保証される．

BSP の特徴の 1 つは，デッドロックがないことである．BSP では，送受信される値を同一スーパーステップ内の計算に用いることができない．そのため，各スーパーステップ内において送受信される値に依存関係がなく，デッドロックが起きないことが保証される．

BSP のもう 1 つの特徴は，リーズナブルな並列

計算機モデルとそのパラメータにより，より実用的に時間計算コストの見積もりができることである．BSP の計算機モデルでは，プロセッサ数 P に加えて，計算速度に対する相対的な通信速度を示す 2 つのパラメータ g と l がある．ここで，g は値 1 つを送信するのにかかる時間であり，l はバリア同期にかかる時間である☆8．あるアルゴリズムが S 個のスーパーステップからなり，i 番目のスーパーステップのプロセッサ p における計算コストが w_p^i，送受信されるデータの量が h_p^i であるとする．そのとき，全体の時間計算コストは

$$\sum_{i=1}^{S}\left(\max_p w_p^i + \max_p h_p^i \cdot g + l\right)$$

として見積もられる．

主に 1990 年代に，BSP や類似の計算モデルの上での並列アルゴリズムが研究され，さまざまな問題に対する BSP アルゴリズムが提案されている [4]．

❑ BSP に基づくマルチコア並列プログラミング

BSP では，その計算モデルに基づいて並列プログラミングを行うためのライブラリ標準 BSPlib が 1996 年に定められた．本章では，BSPlib をサポートするライブラリのうち，近年開発が進められている“MulticoreBSP for C”を用いたマルチコア並列プログラミングについて，実際のコードを用いて紹介する．なお，“MulticoreBSP for C”は，開発元からファイルをダウンロードし，アーカイブの解凍と make によるビルドによりセットアップが完了する☆9．

ここでは例として，接頭和 (prefix sums ; scan) と呼ばれる問題を扱う．接頭和は，入力として配列をとり，前から順に結合的演算子 $\oplus$ により和を求め，その結果を配列に格納する．

$$scan_{\oplus}([a_1, a_2, \ldots, a_n]) = [a_1, a_1 \oplus a_2, \ldots, a_1 \oplus a_2 \oplus \cdots \oplus a_n]$$

たとえば，入力 [3, −1, 4, 1, 5, 9, 2, 6, 5] が与えられたとき，演算子 + を用いて接頭和を計算すると，その結果は [3, 2, 6, 7, 12, 21, 23, 29, 34] となる．接頭和問題は，線形漸化式の並列評価や並列クイックソートの部分処理など，多くの局面で用いられる重要な問題である [3]．

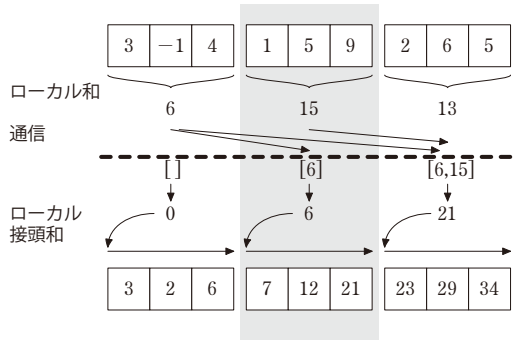

図 -4　BSP による接頭和アルゴリズム

BSP アルゴリズムを設計する際には，一般に，スーパーステップの数が定数かつ少なくなるようにすることが多い．接頭和問題を解く BSP アルゴリズムの 1 つは，**図 -4** に示すように 2 つのスーパーステップからなる．入力には，もとの配列を均等にブロック分割したものを与える．まず，1 つ目のスーパーステップにおいて，各プロセッサは担当範囲の数の総和を求め，求めた和を自分より右を担当するプロセッサへ送信する．次に，2 つ目のスーパーステップにおいて，その前のスーパーステップで受信した値を使って担当範囲より左にある要素の和を求め，その値から始めて自分の担当範囲の接頭和の結果を求める．

この BSP アルゴリズムを“MulticoreBSP for C”を用いて記述したコードの断片を**図 -5** に示す．

“MulticoreBSP for C”では，プログラムは関数を単位として並列に実行され，その関数を実行する前に `bsp_init` 関数を呼ぶ．プログラム例では，並列実行する関数 `parallel` を `bsp_init` によって指定している（48 行目）．一方，並列実行される関数は，並列実行するスレッド数を引数にとる `bsp_begin`（11 行目）から始まり，`bsp_end`（42 行目）で終わる．

“MulticoreBSP for C”を用いたプログラムは，

☆8 L.G.Valiant の論文 [1] では，時間間隔 L で同期を行うような計算機が想定されていた．その場合 $l < L$ である．

☆9 開発元：http://www.multicorebsp.com/．筆者は CentOS 上で実行したが，Windows や Mac OS X でも動作するようである．

```
 1:  #include <mcbsp.h>
 2:  #define SIZE 958003200 /* 2 * 12! */
 3:  #define intS sizeof(int)
 4:
 5:  int *array, *result; /* スレッド間で共有 */
 6:
 7:  void parallel( ) {
 8:    unsigned int i, p, s_i, e_i;
 9:    int acc = 0, *buf;
10:
11:    bsp_begin( bsp_nprocs() );
12:
13:    /* 第0スーパーステップ: 準備 */
14:    s_i = SIZE / bsp_nprocs() * bsp_pid();
15:    e_i = SIZE / bsp_nprocs() * (bsp_pid()+1);
16:    buf = (int*)malloc(intS * bsp_nprocs());
17:    bsp_push_reg(buf, intS * bsp_nprocs());
18:
19:    bsp_sync();  /* バリア同期 */
20:
21:    /* 第1スーパーステップ: ローカルの和を計算 */
22:    for (i = s_i; i < e_i; i++) {
23:      acc += array[i];
24:    }
25:    for (p = bsp_pid()+1; p < bsp_nprocs(); p++) {
26:      bsp_put(p, &acc, buf, bsp_pid()*intS, intS);
27:    }
28:
29:    bsp_sync(); /* バリア同期 */
30:
31:    /* 第2スーパーステップ: 受信データを用いて更新 */
32:    acc = 0;
33:    for (p = 0; p < bsp_pid(); p++) {
34:      acc += buf[p];
35:    }
36:    for (i = s_i; i < e_i; i++) {
37:      acc += array[i]; result[i] = acc;
38:    }
39:
40:    printf("time = %f\n", bsp_time());
41:    bsp_pop_reg( buf );
42:    bsp_end();
43:  }
44:
45:  int main( int argc, char **argv ) {
46:    /* 省略 */
47:    bsp_init( &parallel, argc, argv );
48:    parallel();
49:    /* 省略 */
50:  }
```

図 -5　MulticoreBSP for C を用いた接頭和のコード

スレッドを用いて並列に動作するため，グローバル変数などはスレッド間で共有される（5 行目）．一方，並列実行される関数の中のローカル変数は，スレッドごとに確保されるため独立している．BSP モデル上で送受信されるデータについては，bsp_push_reg 関数を用いて，その領域を登録しなければならない．プログラム例では，スレッド数分の大きさを持つ buf を登録することにより（17 行目），それを後に説明するデータの送信の宛先として使用することができるようになる．

プログラム例の parallel 関数は，bsp_sync によって 3 つの部分に区切られ，データ領域の準備の後，接頭和の計算を行う 2 つのスーパーステップが続く．1 つ目のスーパーステップでは，まず，自分の担当範囲（s_i から e_i まで）の和を計算する．次に，bsp_put 関数を用いて，自分よりも右を担当するスレッドの buf に計算した和を送信している．続く bsp_sync 関数（29 行目）により，バリア同期が実行され，データ送信が完了することが保証される．2 つ目のスーパーステップでは，まず，buf に送られてきた値の和を計算することにより，自分が担当範囲より左の値の総和を計算する．その後，その値を初期値として自分の担当範囲の結果を求めている．

“MulticoreBSP for C”を用いたプログラミングでは，複数のスレッドの動作を明示的に考えて記述することが必要である．その点では，MPI（Message Passing Interface）を用いたプログラミングに近いと言える．しかし，プロセスとして実行される MPI と異なりグローバル変数を共有メモリのように使うことができる点，および，送受信の組を明示的に記述する MPI と異なり送信のみ（または受信のみ）を記述するスタイルで通信が記述できる点の 2 点で，MPI よりもプログラミングがしやすい．

図 -5 のプログラムのパフォーマンスについて簡単に報告する．実験に使用した計算機は，Intel® Xeon® E5645 を 2 つ，メモリ 12GB を持つ計算機である．この計算機において，1 つの for 文によって接頭和を計算する逐次プログラムと，図 -5 の BSP 並列プログラムの実行時間をプロットしたものを**図 -6** に示す．ただし，計算時間は，関数の実行開始からの経過時間を取得する bsp_time 関数（40 行目）によって取得し，並列プログラムではスレッドごとに得られた時間のうちの最大値とする．

BSP 並列プログラムでは，その計算において配列を 2 度読んでいるため，1 スレッドで実行したときには逐次プログラムよりも遅い．しかし，4 スレッドまではきれいな速度向上が得られ，10 スレッドでは並列プログラムの 1 スレッド実行に対して 8.1 倍，逐次プログラムに対して 3.6 倍の速度向上

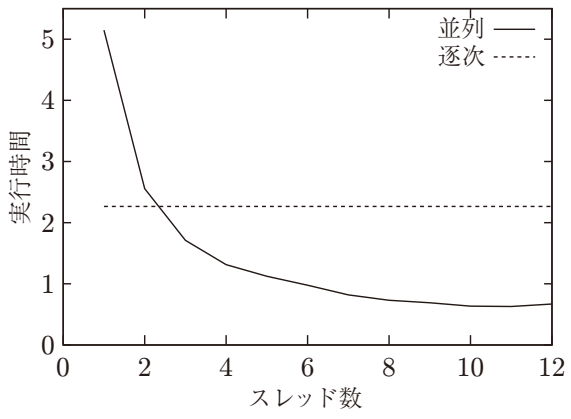

図 -6　接頭和の実行時間

が得られている．

❑ BSP に基づく大規模グラフ処理フレームワーク

冒頭で述べたように，Google の提唱した Pregel は BSP モデルに倣う大規模グラフ処理フレームワークである．Pregel の計算モデルは BSP の計算モデルを基礎とし，ただしプロセッサが主体となって計算・通信するのではなく頂点が主体となって計算・通信する形態となっている．すなわち，Pregel での計算は BSP での計算と同様にスーパーステップの繰り返しで構成され，各スーパーステップでは，各々の頂点が独立に，自身の持つ情報と前のスーパーステップで受信した情報とを元にローカルで計算を行い，自身の持つ情報の更新と他頂点への情報の送信を行う．各頂点に送信される情報は次のスーパーステップで利用可能となる．Pregel モデルは，各頂点にプロセッサとメモリがあるという仮想的な BSP 計算機を考える枠組であるとも言える．現在，Pregel モデルのオープンソース実装として GPS や Giraph が公開されている．以下では，グラフの最短経路問題に関して GPS のサンプルコードを示しつつ Pregel モデルでのプログラミングを紹介する．

有向グラフ $G=(V, E)$ 上の指定された頂点 $v_0 \in V$ からすべての頂点 $v \in V$ への最短経路長 $\mathrm{dist}_0(v)$ を求める，最短経路問題を解く計算を Pregel モデルで考えよう．以下，グラフの各辺 $(v, u) \in E$ に対してその長さを $w(v, u) \in \mathbb{Z}$ とし，頂点 $v \in V$ の隣接頂点集合を $N(v) = \{u \mid u \in V, (v, u) \in E\}$ とする．ここでは天下り的に，次のような Pregel モデルの計算を考える．各頂点 $v \in V$ は，v_0 から自身への最短経路長の暫定値 $d(v)$ を持つ．この暫定値は，$d(v_0) = 0$ および $v \in V \setminus \{v_0\}$ に対して $d(v) = \infty$ と初期化する．各スーパーステップで各頂点 $v \in V$ は，そのすべての隣接頂点 $u \in N(v)$ へ $d(u)$ の更新候補値 $d(v) + w(v, u)$ を送信する．ただし，最初のスーパーステップ以外では，この送信の前に自身の暫定値 $d(v)$ を 1 つ前のスーパーステップで受信した候補値との最小値で更新しておく．いずれの頂点でも暫定値の変化が起こらなければ計算を終了する．なお，自身の暫定値が変化しなかったスーパーステップでは更新候補値の送信の必要がなく，効率のため実装ではこの送信を省くことを注意する．

図 -7 に上記の計算の例を示す．各頂点の中の数は暫定値 $d(v)$ であり，太い辺と数は更新候補値とその送信を意味する．初回のスーパーステップでは，v_0 のみが更新候補値の送信を行っている．次のスーパーステップでは，頂点 v_1 と v_3 が受信した候補値で暫定値を更新し，更新された値に基づき隣接頂点へと更新候補値を送信している．次の 3 回目のスーパーステップも同様の計算を行うが，v_3 は v_1 から候補値を受け取ったものの，その値が現状の暫定値 $d(v_3)$ より大きく $d(v_3)$ が変化しないため，v_3 は候補値の送信を行わない．以降，同様のスーパーステップが繰り返され，5 回目のスーパーステップで暫定値の変化が生じなくなり，最短路長の計算が終了する．実は，この計算はよく知られた Bellman-Ford 法を素直に並列化したものであり，負のサイクルが存在しない限りにおいては必ず計算が終了し，正しく最短路長の値が求められる（すなわちすべての頂点 $v \in V$ で $d(v) = \mathrm{dist}_0(v)$ となる）ことが知られている．

図 -8 に上記の計算を GPS のコードとして素直に記述したものを示す．このコードは，公開されている GPS のサンプルコード（EdgeValueSSSPVertex.java）から，Pregel モデルにおける各スーパーステップでの計算を記述する `compute` メソッドを抜き

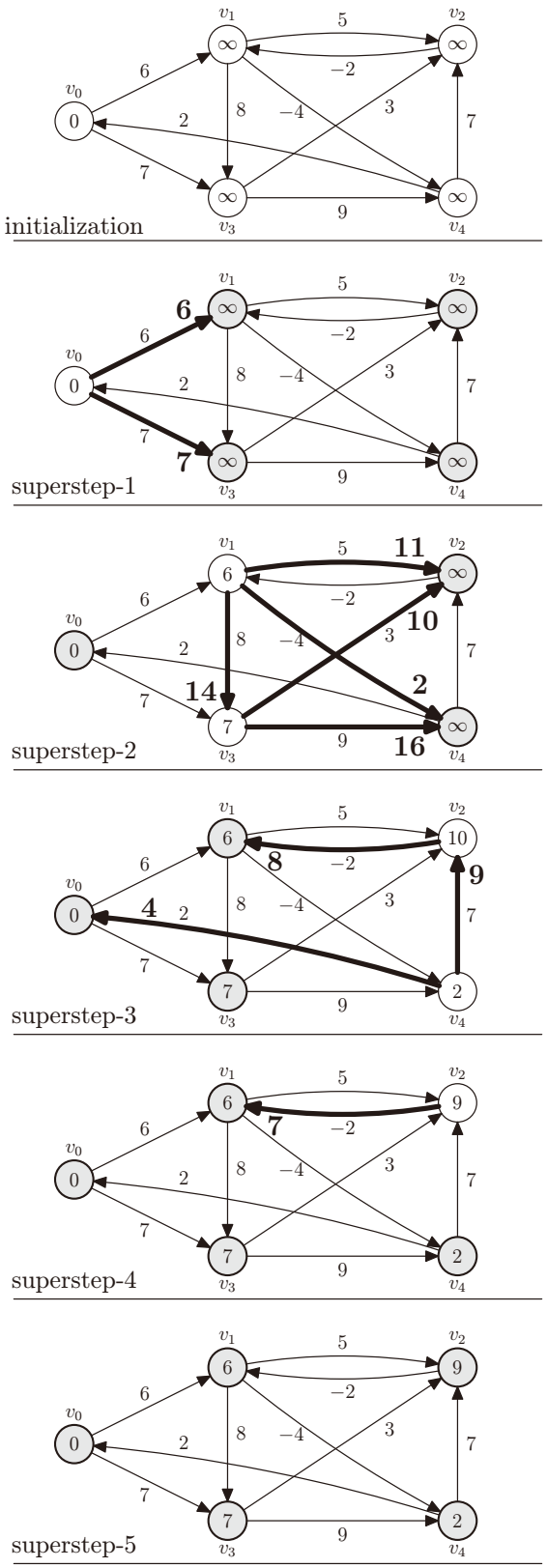

図 -7　Pregel モデルでのグラフ最短路長計算

出し整形したものである．この compute メソッドには，受信したメッセージ（例では IntWritable 型☆10）のイテレータと何回目のスーパーステップであるかの整数値とが渡される（1 行目）．Pregel モデルでは各頂点がローカルに情報を持っており，その情報は getValue メソッドで取得される．このコードでは暫定値 $d(v)$ を IntWritable 型の情報として保持しており，その暫定値の具体的な値は IntWritable##getValue をさらに呼ぶことで取得される（2 行目）．暫定値取得後，compute メソッドは初回の スーパーステップとそれ以降とで大きく分岐する（3 行目）．初回は，暫定値が 0 に初期化されている頂点（すなわち v_0）のみ後述する自作メソッド sendMessageToNeighbors を用いて他の頂点への更新候補値送信を行い（4 行目），それ以外は voteToHalt を実行する（5 行目）．この voteToHalt を実行した頂点は何らかのメッセージを受信するまで以降の スーパーステップの計算をスキップし，すべての頂点が voteToHalt を実行している状態になったスーパーステップで全体の計算が終了する．2 回目以降のスーパーステップでは，各頂点が受信した更新候補値で自身の新しい暫定値を計算し（7-9 行目），より小さい暫定値が得られた場合には隣接頂点へと更新候補値を送信し（10-12 行目），そうでなければ voteToHalt を実行して計算終了に備える（13 行目）．更新候補値の送信をするメソッド（16-20 行目）は，自身を起点とする辺の集合を得る getOutgoingEdges を用いて辺の重みと隣接頂点の ID の取得を行い（17 行目），ID の分かっている頂点に情報を送信する sendMessage を用いて暫定値の送信をしている（18-19 行目）．Pregel モデルでは，各辺の情報はその起点となる頂点で管理されており，また，メッセージの送信先は頂点 ID で指定される．ゆえに，辺に沿った情報の送信が基本となる．

以上のように，Pregel モデルではグラフ計算の為の並列プログラムをデッドロック等の心配なしに容易に記述することができる．ただし，Pregel モ

☆10 Hadoop における整数型 int のラッパー型．

```
 1:  public void compute(Iterable<IntWritable> messageValues, int superstepNo) {
 2:    int previousDistance = getValue().getValue();          // 現在の最短路長暫定値を取得
 3:    if (superstepNo == 1) {                                // 初回スーパーステップでは
 4:      if (previousDistance == 0) sendMessageToNeighbors(); //   ルートのみ隣接頂点へ更新候補値を送信
 5:      else                       voteToHalt();             //   他は停止投票
 6:    } else {                                               // 2回目以降のスーパーステップでは
 7:      int minValue = previousDistance;
 8:      for (IntWritable messageValue : messageValues)       //   受信した更新候補値で新しい暫定値を計算
 9:        minValue = Math.min(minValue, messageValue.getValue());
10:      if (minValue < previousDistance) {                   //   暫定値が小さくなるならば
11:        setValue(new IntWritable(minValue));               //     暫定値を更新し
12:        sendMessageToNeighbors();                          //     隣接頂点へと更新候補値を送信
13:      } else { voteToHalt(); }                             //   暫定値の変更がなければ停止投票
14:    }
15:  }
16:  private void sendMessageToNeighbors() {                  // 隣接頂点への更新候補値送信
17:    for (Edge<IntWritable> outgoingEdge : getOutgoingEdges())
18:      sendMessage(outgoingEdge.getNeighborId(),            // メッセージ送信先は ID で指定
19:          new IntWritable(getValue().getValue()+outgoingEdge.getEdgeValue().getValue()));
20:  }
```

図 -8　最短路長の計算を行う Pregel（GPS）プログラムの主要部

デルは頂点が主体となる計算モデルのため，辺を主体として計算を行うよう設計された一部のグラフ処理アルゴリズムを Pregel モデルに直接持ち込むのは難しい．この困難解決のために，領域特化言語からの Pregel プログラムへの変換も研究されている．また，GPS はネットワーク I/O 削減のための最適化などにも力を入れている．

❑ BSP に関連する研究動向・今後の期待

近年，マルチコア CPU を持つ計算機が普及し，また GPGPU などアクセラレータの利用も行われている．このようなハードウェア環境の複雑化や，その上での高水準プログラミングへの要求から，BSP に基づくまたは BSP を拡張する研究がいくつかの研究グループによって行われている．その 1 つは，BSP の提案者 L.G.Varliant による Multi-BSP である [2]．Multi-BSP では，多段階の階層を持つ計算機モデルが考えられており，簡単に言うと図 -2 の BSP 計算機モデルのプロセッサの部分に，BSP の計算機モデルが入ったようなモデルである．前に紹介した“MulticoreBSP for C”は，実は，そのような Multi-BSP モデルに基づくプログラミングも扱うことができるように設計されており，標準の BSPlib からいくつか拡張されている．筆者らは，BSP モデルに基づいて設計したアルゴリズムを MapReduce 上で実行する方法についても研究を行っている．

冒頭で述べたように，Google によって提案された大規模グラフ処理フレームワーク Pregel によって BSP が再び注目されるようになった．しかし，Pregel などが利用しているのはスーパーステップに基づく BSP の計算モデルのみである．BSP の元論文が述べるように，並列計算におけるハードウェアとソフトウェアのギャップを埋めることは重要であるものの，そのような“Bridging Model”に関する研究はまだ十分とは言えない．今後，その分野の研究が進展することを期待して，この記事のまとめとしたい．

参考文献

1) Valiant, L. G. : A Bridging Model for Parallel Computation, Communications of the ACM, Vol.33, No.8 (1990).
2) Valiant, L. G. : A Bridging Model for Multi-core Computing, Journal of Computer and System Sciences, Vol.77, No.1, pp.154-166 (2011).
3) Blelloch, G. E. : Prefix Sums and their Applications, Technical Report CMU-CS-90-190, School of Computer Science, Carnegie Mellon University (1990).
4) Frank, K. H., Dehne, A., Ferreira, A., Cáceres, E., Song, S. W. and Roncato, A. : Efficient Parallel Graph Algorithms for Coarse-grained Multicomputers and BSP, Algorithmica, Vol.33, No.2, pp.183-200 (2002).
5) Bilardi, G., Herley, K. T., Pietracaprina, A., Pucci, G. and Spirakis, P. : BSP vs logP, SPAA '96 : Proceedings of the Eighth Annual ACM Symposium on Parallel Algorithms and Architectures, pp.25-32 (1996).

（2015 年 1 月 27 日受付）

松崎 公紀（正会員）matsuzaki.kiminori@kochi-tech.ac.jp

高知工科大学准教授（2009 年 10 月より），博士（情報理工学）．高水準並列プログラミング手法やアルゴリズム導出のほか，ゲーム・パズルプログラミングなどにも興味を持つ．日本ソフトウェア科学会，ACM，IEEE 各会員．

江本 健斗（正会員）emoto@ai.kyutech.ac.jp

九州工業大学准教授（2015 年 4 月より），博士（情報理工学）．高水準並列プログラミング手法，アルゴリズム導出，それらの定理証明支援系による形式的証明などに興味を持つ．日本ソフトウェア科学会，ACM 各会員．