Xウインドウを用いたハイパーキューブの並列アルゴリズム表現

塩田佳明, 渋沢進

茨城大学工学部情報工学科

本研究では, Xウインドウ上において反射2進順にレイアウトしたハイパーキューブを表現し, その上で3種類の基本的な並列アルゴリズムの実行を表現するシミュレータを作成した. 反射2進順にレイアウトすることで, プロセッサ数が大きいときは昇順にレイアウトした場合の約4/9の面積で実現できる. また, ハイパーキューブアルゴリズムだけでなく格子結合のアルゴリズムも実行可能である.

並列アルゴリズムとしては, バイトニックソート, プレフィクス計算, および行列積を取り上げた. プログラムにおいて, グラフィック部分とアルゴリズム部分はほぼ独立しており, 別のアルゴリズムの表現も比較的容易である. 作成したシミュレータにより, 反射2進ハイパーキューブ結合モデル上で各種並列アルゴリズムが実行される様子, およびそれに伴うプロセッサ間のデータ転送, 比較交換等を視覚的に理解することが可能となり, 教育用ツールとして十分耐えられるものとなっている.

An expression of parallel hypercube algorithms using X-Window

Yoshiaki Shiota, Susumu Shibusawa

Faculty of Engineering, Ibaraki University

In this research, we express a binary-reflected hypercube with the X-Window, on which we build three simulators which express basic parallel algorithms. The binary-reflected hypercube only takes about 4/9 of the space of binary hypercube for the large number of nodes. Besides, the binary-reflected hypercube can also perform mesh algorithms.

We treated three parallel algorithms which are bitonic sort, prefix computation, and matrix multiplication. In these programs, graphic routines and algorithm routines are nearly independent, thus it becomes easier to express other algorithms. With the simulators, it is possible to understand visually parallel algorithms, data transmission, and comparison and exchange of data on the binary-reflected hypercube model. Thus it will be an useful tool in the field of education for computer architecture and algorithms.

1 はじめに

これまでに, 多数の並列計算機アーキテクチャが考案されてきた [1]-[3]. このうち, ハイパーキューブ結合は, 隣接ノードと通信する局所結合であり, 木結合, 格子結合等を含んでいるため, 有力なアーキテクチャの1つであると考えられている.

また, 幅優先探索などの各種グラフ, ソーティングアルゴリズム等の動きを視覚的に表現してアルゴリズムの理解, アルゴリズムの評価, 新たなアルゴリズムの発達を促すようなシミュレータ (アニメーション) が作られている [4] [5].

本研究では, 並列処理におけるプロセッサ間結合, および, その上で動作する並列アルゴリズムの実行の過程を視覚的に表現し, 理解しやすくすることで, 教育用ツールとしての役割と同時に, 計算機やソフトウェアの開発設計支援に役立つであろうと考え, シミュレータを作成した.

プロセッサ間結合としてハイパーキューブ結合を取り上げる. 反射2進順にレイアウトしたハイパーキューブでは, 結合の一部はすべてのプロセッサに対して明確な2次元格子結合を形成し, ハイパーキューブアルゴリズムだけでなく, 格子結合アルゴリズムも実行可能である. さらに, プロセッサ数が大きいときは, 昇順ハイーパーキューブの約 4/9 のレイアウト面積で実現できる [6].

シミュレータの実行環境としては, 現在 UNIX 上の標準的な GUI である X ウインドウを用いることにした. X のライブラリは Xlib のみを用いている. 具体的な並列アルゴリズムとしては, 行列積, プレフィクス計算, および, 代表的な並列ソーティングアルゴリズムの1つであるバイトニックソートを取り上げた. これらのアルゴリズムはすべて基本的な問題であり, n データのプレフィクス計算とバイトニックソートは n プロセッサの昇順と反射2進ハイパーキューブ上でそれぞれ $O(\log n), O(\log^2 n)$ ステップで実行でき, $n \times n$ 次の2行列の積は n^3 プロセッサをもつハイパーキューブにおいて $O(\log n)$ ステップで実行できる.

シミュレータの作成においては, まず反射2進ハイパーキューブ結合モデルを X ウインドウの正方形, 直線という基本図形で表現し, さらにアルゴリズム表現に必要なデータの通信およびデータを描画する手続きを作成した. この上にハイパーキューブ向き各種並列アルゴリズムの手続きを構築している. アルゴリズム表現のための基本的な描画手続きを作成したことで, 新たに別のアルゴリズムを表現することも比較的容易である.

作成したシミュレータにより, 反射2進ハイパーキューブ結合モデル, 各種並列アルゴリズムがそれぞれのオーダで実行される様子, およびそれに伴うプロセッサ間のデータ転送, 比較交換等を視覚的に理解することが可能である.

2 準備

2.1 反射2進グレイ符号

引き続く符号語が1ビットだけ異なるような巡回的系列をグレイ符号と呼び, このうち符号語が2のべき乗ごとに反射的に1ビットずつ変化するグレイ符号が反射2進グレイ符号である. これは, 次のようにして求める. ある非負整数に対する m ビット2進符号語 e と反射2進グレイ符号語 u を以下のように2進表現する.

$$e = [e_{m-1} \cdots e_1 e_0],\ u = [u_{m-1} \cdots u_1 u_0] \quad (1)$$

$$(e_i, u_i \in \{0,1\}, 0 \le i \le m-1)$$

このとき, 反射2進符号 u の第 i 桁 u_iは, 2進符号語 e の第 $i, i+1$ 桁 e_i, e_{i+1}を用いて, 次のように表される.

$$e_i = \begin{cases} e_i \oplus e_{i+1} & (0 \le i \le m-2) \\ e_{m-1} & (i = m-1) \end{cases} \quad (2)$$

ここで, 記号$\oplus$は排他的論理和を表す. また, 2進符号 e の第 i 桁 e_iは反射2進符号 u を用いて次のように表される.

$$e_i = \bigoplus_{j=i}^{m-1} u_i \quad (0 \le i \le m-1) \quad (3)$$

2.2 下位ビット演算

ハイパーキューブのプロセッサを反射2進順にレイアウトするための表記法を導入する [7].

[定義1] (1) 非負整数 e の下位 i 桁 (lower i bits) を記号 LBで表す. e が式 (1) の 第一式で表されるとき,

$$LB(e,i) = [e_{i-1} \cdots e_1 e_0] \quad (4)$$

ただし, $i=0$ のとき, $LB(e,0)=0$ とする.
(2) 整数 e の2進表現の第 i 桁 $e_i=0$ ならば, e の2進表現の下位 i 桁に等しく, $e_i=1$ ならば, e の補数 $\bar{e}$ の下位 i 桁に等しい表現を記号 $LBr(e,i)$ で表す.

$$LBr(e,i)=\begin{cases} LB(e,i) & (e_i=0) \\ LB(\bar{e},i) & (e_i=1) \end{cases} \tag{5}$$

ただし, $i=0$ のとき, $LBr(e,0)=0$ とする. また, $e_i=0$ のとき $LB(\bar{e},i)$ に等しく, $e_i=1$ のとき $LB(e,i)$ に等しい表現を記号 $LBc(e,i)$ で表す.

$$LBc(e,i)=\begin{cases} LB(\bar{e},i) & (e_i=0) \\ LB(e,i) & (e_i=1) \end{cases} \tag{6}$$

ただし, $i=0$ のとき, $LBc(e,0)=0$ とする.
[補題1] 非負整数 e に関して, 以下の関係が成り立つ.

(1) $LBr(e,i)=\begin{cases} LB(e,i+1) & (e_i=0) \\ LB(\bar{e},i+1) & (e_i=1) \end{cases}$

(2) $\begin{array}{rcccl} 0 & \leq & LB(e,i), LB(\bar{e},i) & \leq & 2^i-1 \\ 0 & \leq & LBr(e,i), LBc(e,i) & \leq & 2^i-1 \end{array}$

(3) $LB(e,i)+LB(\bar{e},i)=2^i-1$

(4) $LBr(e,i)+LBc(e,i)=2^i-1$

(5) $[e_i e_{i-1}]=[01]$ または $[10]$ のとき,
$LBr(e,i)+LBr(e,i-1)=2^j-1$
$LBc(e,i)+LBc(e,i-1)=2^{j-1}-1$

3 ハイパーキューブのレイアウト

3.1 1次元レイアウト

3.1.1 昇順レイアウト

正整数 m に対して, 1次元方向に $n=2^m$ 個の配列化したノード $e(I)(0\leq I\leq 2^m-1)$ を置く. 配列 I を2進符号化して, これを e^x とおく. ここで

$$I=e^x=[e_{m-1}\cdots e_1 e_0] \tag{7}$$
$$(e_i\in\{0,1\}, 0\leq i\leq m-1)$$

式(7)で表される2ノード間にハイパーキューブ結合を導入する. 2ノードの2進表現のハミング距離が1のときに, これら2ノードは結合される. 図1は, $n=2^3$ のハイパーキューブの1次元レイアウト例である. 図中の□はノードを表し, 実線はキューブ結合における通信リンクを表す.

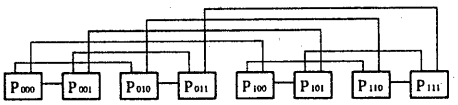

図1: 昇順レイアウト

配線幅を単位長さとするレイアウトのモデルに従い, この昇順1次元レイアウトハイパーキューブのレイアウト面積を評価する. 図1のレイアウトに対して, ノードの横方向の一辺は $(m-1)$ 端子を持つので, 横方向の長さが $(m-1)2^m$ で実現できる. また, ハイパーキューブ結合による配線領域の幅が $(2+2^2+\cdots+2^{m-1})$ であり, ノードの縦の長さはこれより小さく実現できるので, レイアウト面積は以下のようになる.
[補題2] n ノードハイパーキューブの昇順1次元レイアウトの面積 $A_{bin}^{(1)}$ は,

$$\begin{aligned} A_{bin}^{(1)} &= (m-1)2^m\times[2+2^2+\cdots+2^{m-1}] \\ &= n^2\log n+O(n^2) \end{aligned} \tag{8}$$

3.1.2 反射2進順レイアウト

正整数 m に対して, 1次元方向に $n=2^m$ 個の配列化したノード $u(I)(0\leq I\leq 2^m-1)$ を置く. 配列 I を反射2進符号化し, これを u^x とし, 次のように2進表現する.

$$u^x=[u_{m-1}\cdots u_1 u_0]\,(u_i\in\{0,1\}, 0\leq i\leq m-1) \tag{9}$$

昇順レイアウトと同様に, ノード間にハイパーキューブ結合を導入する. 図2は, $n=2^3$ の反射2進ハイパーキューブの1次元レイアウト例である. 図の表し方は昇順のものと同様である. 上位桁からレイアウトする場合と, 下位桁からレイアウトする場合があるが, 図2は上位桁から2ビットずつ対にレイアウトしたものである. 以後, 上位桁から2ビットずつ対にするレイアウトを取り扱う.

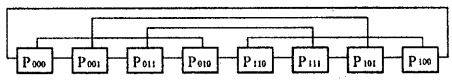

図2: 反射2進順レイアウト

次に, このレイアウト面積を評価する. ノードの横方向の一辺は $(m-2)$ 端子を持つので, レイアウト

の横方向は長さ $(m-2)2^m$ で実現できる．また，レイアウトの縦方向の配線幅は通信リンクの高さ $y^{(i)}$ の最大値で与えられる．

[補題3] 上位桁からレイアウトするときの第 i 桁の通信リンクの高さ $y^{(i)}$ は以下のようになる．次元が偶数のときは，

$$y^{(i)} = \begin{cases} [\{10\}^{\frac{i}{2}}] + \frac{i}{2} + LBr(x,i) - 1 & (i:\text{偶数}) \\ [\{10\}^{\frac{i-1}{2}}] + \frac{i-1}{2} + LBc(x,i) & (i:\text{奇数}) \end{cases}$$

次元が奇数のときは，

$$y^{(i)} = \begin{cases} [\{10\}^{\frac{i}{2}-1}0] + \frac{i}{2} + LBc(x,i) & (i:\text{偶数}) \\ [\{10\}^{\frac{i-1}{2}}0] + \frac{i-1}{2} + LBr(x,i) & (i:\text{奇数}) \end{cases}$$

ここで，x はプロセッサの水平方向の位置，i は u^x の第 i 桁である．また，

$$[\{10\}^i] = [\underbrace{1010\cdots10}_{2i\text{ビット}}]$$

ただし，次元の偶奇に関わらず，$y^{(1)} = 1$，物理的に隣合うノードを結ぶ通信リンクの高さは0とする．

$y^{(i)}$ の最大値は，

$$\frac{2}{3}2^m + O(m)$$

となり，各ノードの縦の長さは配線幅よりも小さく実現できるので，レイアウト面積は以下のようになる．

[補題4] n ノードハイパーキューブの反射2進1次元レイアウトの面積 $A_{ref}^{(1)}$ は，

$$\begin{aligned} A_{ref}^{(1)} &= (m-2)2^m(\tfrac{2}{3}2^m + O(m)) \\ &= \tfrac{2}{3}n^2\log n + O(n^2) \end{aligned} \quad (10)$$

[定理1] ハイパーキューブのノード数が大きいとき，昇順と反射2進順の1次元レイアウトの面積 $A_{bin}^{(1)}$ と $A_{ref}^{(1)}$ の間には，次の関係が成り立つ．

$$A_{ref}^{(1)} \simeq \frac{2}{3}A_{bin}^{(1)} \quad (11)$$

このことは，反射2進1次元レイアウトは，昇順1次元レイアウトに比べて，約 2/3 の面積で実現できることを意味する．

3.2 2次元レイアウト

3.2.1 昇順レイアウト

正整数 m に対して，2次元平面 xy 上の配列 $I, J(0 \le I, J \le 2^m - 1)$ に $n = 2^m \times 2^m$ 個のノードを置く．配列 I, J をそれぞれ座標 x, y 方向にとり，配列 I, J の2進符号をそれぞれ e^x, e^y とおく．

$$\begin{aligned} I &= e^x = [e_{m-1}\cdots e_1 e_0], \\ J &= e^y = [e_{2m-1}\cdots e_{m+1}e_m] \end{aligned} \quad (12)$$

$$(e_i \in \{0,1\}, 0 \le i \le 2m-1)$$

配列 I, J にあるノードを $e(I,J) = \langle e^x, e^y\rangle$ で表せば，

$$e(I,J) = \langle e^x, e^y\rangle = [e_{2m-1}\cdots e_1 e_0] \quad (13)$$

式 (13) で表される2ノード間にハイパーキューブ結合を導入する．結合は，x, y の両方向にまたがることはない．図3は，$n = 2^6 (m = 3)$ の昇順ハイパーキューブの2次元レイアウト例である．

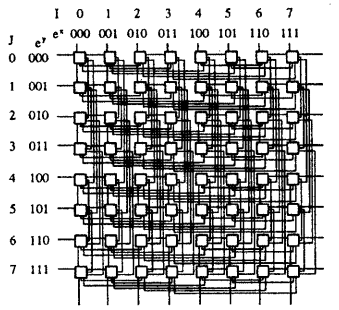

図3: 昇順2次元レイアウト

このレイアウトの面積を評価する．1次元レイアウトの例から，この2次元レイアウトの一辺の長さは，高々

$$\begin{aligned} &2^m \times [(2 + 2^2 + \cdots + 2^{m-1}) + O(m^c)] \\ &= n + O(\sqrt{n}\log^c n) \end{aligned} \quad (14)$$

ここに c は定数である．これより，レイアウト面積は，次のように表される．

[補題5] ある正整数 m に対して，$n = 2^m \times 2^m$ ノード昇順ハイパーキューブの2次元レイアウト面積 $A_{bin}^{(2)}$ は次のように表される．

$$A_{bin}^{(2)} = n^2 + O(n^{3/2}\log^c n) \quad (15)$$

3.2.2 反射2進順レイアウト

正整数 m に対して，2次元平面 xy 上の配列 $I, J(0 \le I, J \le 2^m - 1)$ に $n = 2^m \times 2^m$ 個のノードを置く．

配列 I, J の反射2進符号をそれぞれ u^x, u^y とし，これらを次のように2進表現する．

$$u^x = [u_{m-1} \cdots u_1 u_0], \quad u^y = [u_{2m-1} \cdots u_{m+1} u_m] \tag{16}$$

$$(u_i \in \{0,1\}, 0 \leq i \leq 2m-1)$$

配列 I, J にあるノードを $u(I,J) = \langle u^x, u^y \rangle$ で表せば，

$$u(I,J) = \langle u^x, u^y \rangle = [u_{2m-1} \cdots u_1 u_0] \tag{17}$$

図4は，$n = 2^6 (m = 3)$ の反射2進ハイパーキューブの2次元レイアウト例である．また，図4はハイパーキューブ結合を上位桁から2ビットずつ対にしてレイアウトしたものである．

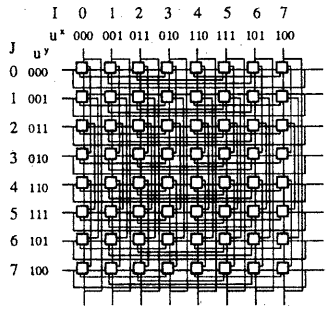

図4: 反射2進順レイアウト

このレイアウト面積は，以下のようになる．

[定理2] ある正整数 m に対して，$n = 2^m \times 2^m$ ノード反射2進ハイパーキューブのレイアウト面積 $A^{(2)}_{ref}$ は，各方向とも上位から2ビットずつ対にしてレイアウトするとき，次のように表される．

$$A^{(2)}_{ref} = 4n^2/9 + O(n^{3/2} \log^c n) \tag{18}$$

ここに c は定数を表す．

[定理3] $n = 2^m \times 2^m$ ノードハイパーキューブの昇順と反射2進順の2次元レイアウトの面積 $A^{(2)}_{bin}, A^{(2)}_{ref}$ の間には，反射2進ハイパーキューブのレイアウトを上位ビットから対にし，ノード数が大きいとき次の関係がある．

$$A^{(2)}_{ref} \simeq 4A^{(2)}_{bin}/9 \tag{19}$$

定理より，ノード数が大きいとき，反射2進ハイパーキューブのレイアウト面積は，昇順ハイパーキューブの約 4/9 の面積で実現できる．

4 X ウインドウを用いたハイパーキューブの表現

まずアルゴリズムを表現する前に，X ウインドウ上での反射2進ハイパーキューブの表現，および実現方法を示す．後述の各種アルゴリズムも，この表現の上に構築されている．なお，シミュレータプログラムは，C言語で記述し，X プログラミングのライブラリは Xlib を用いた．

プログラムの構成は図5のように大きく3つの部分に分けられる．下側の2つの部分により，X ウインドウ上に反射2進ハイパーキューブが描かれ，アルゴリズムを表現するための部分が作られている．この上にアルゴリズム部分を付加することにより，シミュレータが成りたっている．このことにより，アルゴリズム部分を交換してやることによって別のアルゴリズムの表現も比較的容易である．

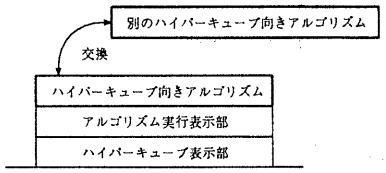

図5: プログラムの構成

反射2進ハイパーキューブの表現に最も重要なのは，各プロセッサを表すデータ構造である．これは，C言語の構造体の配列を以下のように定義した．

```
#define NODE_PARAM 8
#define DIMENSION 3

typedef struct {
    int height;
    int connect;
    int start_x;
    int start_y;
}Line;

typedef struct {
    Line line_x[DIMENSION];
    Line line_y[DIMENSION];
    int rec_x;
    int rec_y;
    int data;
}PROSESSOR;

PROSESSOR  processor[NODE_PARAM][NODE_PARAM];
```

NODE_PARAM, DIMENSION は，それぞれ x, y 方向のノード数，次元である．実際は NODE_PARAM $= 2^m$ (m

= DIMENSION) であるが, 便宜上こうしてある. 最も重要なのが, 通信リンクに関するデータ構造で, それぞれ DIMENSION 本分の通信リンクを Line 型の配列である Line_x, Line_y に確保している. これは, 6 次元ハイパーキューブなら x, y 方向それぞれ 3 本の合計 6 本の通信リンクをもつことを意味する. Line 型構造体は, 図 6 に示すようにその通信リンクの高さおよび, 接続するプロセッサ, プロセッサのどの座標から通信リンクが描かれるかの情報をもっている. rec_x, rec_y はプロセッサを表す正方形を描くための, 正方形の左上の x, y 座標の値を持つ.

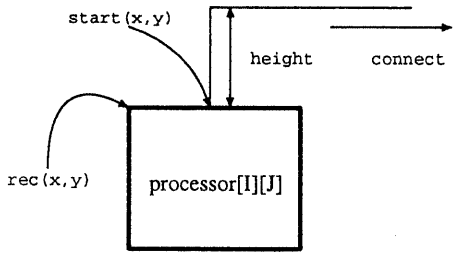

図 6: プロセッサ変数の意味

変数 data はアルゴリズム実行時のデータ保存用にもつことにする. このデータ構造を用いて, 反射 2 進ハイパーキューブを次のような流れで表現した.

1. 反射 2 進ハイパーキューブを描くウインドウの設定.
2. グレイ符号の生成.
3. 各通信リンクがどのプロセッサと接続するのかを計算する.
4. 各通信リンクの高さを求める.
5. プロセッサを描く位置を決める.
6. 各通信リンクを描く位置を計算する. これは, x, y それぞれの方向について計算する.
7. プロセッサ (正方形) を描く.
8. 各通信リンクを描く.

1 は, 描くウインドウの細かい設定を行なう. これは X プログラミングには欠かせないものである.

2 で反射 2 進グレイ符号を生成するのだが, x, y 方向がそれぞれ m 次元のとき, m ビット反射 2 進グレイ符号を作るようにする. これを使って, 後々まで様々な処理をすることになる.

3 で, 各通信リンクがどのプロセッサと接続するのかを計算する. ここでは各プロセッサのグレイ符号のアドレスを 1 ビットずつ反転させて, 反転させたものと一致するプロセッサを探すことで接続プロセッサを求めている. ここでは, すべての通信リンクの高さを計算することは避け, 第 1 行のプロセッサの x 方向の高さを求めて, それをすべての行, すべての列にコピーすることで処理を高速化している.

5 , 6 でウインドウ上のどの位置に描くのかを決定する. 各通信リンクを描く位置の計算は, 各通信リンクがどのプロセッサと接続するのかによって細かい分岐処理を行なっている. またプロセッサは正方形で表されているのであるが, 通信リンクはその 1 辺を等分するように並べる.

7, 8 で, 計算させたデータを使って, 実際にウインドウ上に描画する.

以上の流れで反射 2 進ハイパーキューブモデルを表現した. なお, このモデルに関しては, ある程度任意の次元を表示できるようにしてあるが, ディスプレイの大きさに限界があり, 最大は 8 次元ほどである.

5 X ウインドウを用いた並列アルゴリズムの表現

並列アルゴリズムを表現するといっても, プログラムでは並列に記述することはできない. しかも Sun Sparc-Station の場合, X ウインドウの描画速度はそれ程速くはないが, 幸いなことに, 順序的なプログラムであっても, 並列に動作しているように見せるだけの能力はある. 以下では, ハイパーキューブ上の並列アルゴリズムを X ウインドウ上において, どのようにして表現, 構成, プログラムしたかを述べる. なお, モデルとした反射 2 進ハイパーキューブは, $n = 2^3 \times 2^3$ 個のプロセッサを 2 次元平面上に配置したものを用いている. 理由は, これが一番見やすく, プロセッサ数も比較的多いことによる.

5.1 データ通信の表現

プロセッサ間データ通信の表現は, 通信を行なっている通信リンクをウインドウ上で太く描くことで表現した. プログラム中では, Xlib ライブラリの 描画属性 GC (graphic context) には, 白黒を反転させる機能をもつ論理関数 GXinvert を用いて描画を容易にしている. これの最大の特徴は 2 度同じものを描くと元の描く前の状態に戻ることである. これを用いることで, データ通信の際は, 通信リンクを表して

いる細い線が描いてあったところは白く抜けてしまうものの(図7), データ転送が終れば, もう1度同じ太い通信リンクを描けば元の黒の細い線の状態に戻るので, 都合が良い.

プロセッサが保有するデータは, ウインドウ上のプロセッサの中にデータを書き入れることで表現している. このことにより, データの移動が理解しやすくなっている.

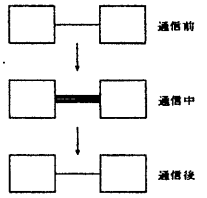

図7: データ通信の表現

5.2 バイトニックソート

ハイパーキューブ上でのバイトニックソートについて, 簡単に説明する. 図8は, バイトニックソートの一表現である. 図において, 水平方向の線はバイトニックソートの過程を表す. 垂直方向の矢印は2データに対する比較器を表しており, 矢印の根元に小さいデータ, 矢印の先に大きいデータが置かれるものとする. 図のバイトニックソートでは, 第i段階$(0 \leq i \leq k-2, k = \log_2 n)$の終りには, 長さ$2^{i+2}$のバイトニック系列が生成されて, 最後の第$(k-1)$段階の終りにはソートされた結果が出力される.

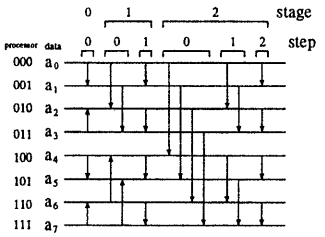

図8: バイトニックソートの一表現

この図8の垂直方向の各位置を, 式(17)で表されるnプロセッサ反射2進ハイパーキューブのプロセッサuに置き換え, 反射2進ハイパーキューブ上でnデータに対するバイトニックソートを実行する. 本研究では反射2進ハイパーキューブ上でのシミュレータを作成したが, 昇順ハイパーキューブ上でも同様である.

バイトニックソートの第i段階のステップjにおける比較前のノードのデータを$c_{ij}(u)$とおく. このときに, プロセッサの2進表現の第$(i-j)$桁の値u_{i-j}のみが異なる隣接プロセッサ間で比較交換操作が行なわれ, この操作を$\mathrm{comp}(c_{ij}(u_{i-j}))$とする. この際, プロセッサの2進表現の第$(i+1)$桁が0の場合は$u$に関してデータを昇順に比較交換し, 1の場合は降順に比較交換する. ここでバイトニックソートの手法を用いている.

ハイパーキューブ上でのバイトニックソートのアルゴリズムは以下のようになる.

```
1  procedure bitonic-sort;
2  for i ← 0 to m-1 do          /* stage */
3     for j ← 0 to i do         /* step */
4        for u ← (0, 0) to (2^(m-1), 2^(m-1)) pardo
5           DrawEvent (c_ij(u_i-j))
5           comp(c_ij(u_i-j)) ;
5           DrawResult
6        odpar
7     od
8  od
```

図9: バイトニックソートを実行する手続き

図9の疑似プログラム中の記号 for ⋯ do ⋯ od は 順序的な操作で, for ⋯ pardo ⋯ odpar は並列的な操作を表している. この疑似プログラムを, 順序的なプログラムに変換している. 実際のプログラムはこれに比べかなり複雑である. 図には, シミュレータの部分も含まれていて, DrawEvent により, 現在通信中のプロセッサ同士を結ぶ通信リンクがが太くなる. DrawResult によって, それぞれのプロセッサ上にに現在のデータが表示される.

5.3 プレフィクス計算

結合法則が成り立つ2項演算を$*$とする. プレフィクス計算(prefix computaion)とは, 与えられたn個のデータ$\{a_0, a_1, \cdots, a_{n-1}\}$に対して, n個の値

$$S_i = a_0 * a_1 * a_2 * \cdots * a_i \,, \; (0 \leq i \leq n-1)$$

を求める問題である[3]. 本研究では, $*$演算として加算を取り上げた. この場合, 各S_iはプレフィクス・サムと呼ばれる. プログラムにおけるグラフィック部分はバイトニックソートと同様である.

5.4 行列積

ハイパーキューブの行列積アルゴリズム [2] では, $n \times n$ 次の 2 行列の積を n^3 個のプロセッサを用い, $O(\log n)$ ステップで実行する. 例えば $2^3 \times 2^3$ プロセッサハイパーキューブでは, $2^2 \times 2^2$ 次の 2 行列の積が実行できる. データ転送にはブロードキャスト操作が必要で, 実行時間のほとんどはデータ転送に要する時間である. やはりグラフィック部分は同様である.

6 実行結果

図 10 は, バイトニックソートの実行中の画面である. 64 個のプロセッサにランダムに入力されたデータをソートする. 反射 2 進順に配置されているため最終結果も反射 2 進順になる. 視覚的表現により, データの流れる様子が理解しやすい.

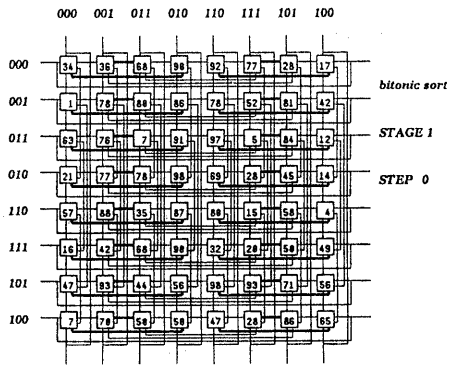

図 10: 実行画面

7 おわりに

ハイパーキューブの特徴, 反射 2 進グレイ符号, 各種並列アルゴリズム, および X ウインドウ上の反射 2 進レイアウトハイパーキューブの表現, アルゴリズムシミュレータの実現法等を示した.

本研究により, ハイパーキューブ上でのアルゴリズムの動き, データ転送の流れを視覚的に理解することができる. アルゴリズムを知らない人でも, シミュレータを見ながら説明を受ければ容易に理解できるようになる. これから並列アルゴリズムを勉強する人, 計算機アーキテクチャやソフトウェアの設計開発をする人にはかなり役立つものになるであろう. また, ハイパーキューブをプログラムで表すために作成したデータ構造 (具体的には, 構造体など), および今回得たノウハウは, 今後同様のソフトウェア開発をする際の作業を軽減するものと思われる. 実際, 別のアルゴリズムを走らせるのはそれほど難しくはなく, アルゴリズム部分の手続きさえ作ってやれば良い. 今回もバイトニックソートを最初に実現し, 他の 2 つはそれほど時間はかかっていない.

今後の課題としては, さらなるユーザインタフェースの強化が挙げられる. また本研究では, アルゴリズムの手続きからグラフィック用の手続きを呼んでいるが, 理想的には, アルゴリズム部分とグラフィック部分が完全に独立して, 並列な疑似コードで書かれたアルゴリズムを入力すると, シミュレータが解釈実行するようなものも考えられる.

参考文献

[1] F.T.Leighton: Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes, Morgan Kaufmann (1992).

[2] 梅尾 博司: 超並列計算機アーキテクチャとそのアルゴリズム, 共立出版 (1991).

[3] 梅尾 博司: SIMD 上の並列アルゴリズム, 情報処理, Vol.33, No.9, pp.1042-1055 (1992).

[4] M.H.Brown: Exploring Algorithms Using Basla-II, IEEE Computer, Vol.21, No.5, pp.14-36 (1988).

[5] J.T.Stasko: Tango: A Framework and System for Algorithm Animation, IEEE Computer, Vol.23, No.9, pp.27-39 (1990).

[6] 渋沢 進: 超立方体結合と大域バス結合より成る一相互結合モデルについて, 情報処理学会論文誌, Vol.32, No.4, pp.532-543 (1991).

[7] 渋沢 進: CCC ネットワークのレイアウトの一評価, 電子情報通信学会論文誌, Vol.J71-D, No.7, pp.1349-1353 (1988).