論　文

# パワーリスト記述からのハイパキューブアルゴリズムの生成

東大野雅之[†]　　西谷　泰昭[††]

Generating Hypercube Algorithms from Powerlist Descriptions

Masayuki HIGASHIOHNO[†] and Yasuaki NISHITANI[††]

**あらまし**　Misra が提案したパワーリストは，並列再帰構造を仕様記述するためのデータ構造である．Achatz and Schulte はパワーリスト関数による仕様記述から，相互結合網非依存の超データ並列アルゴリズムを生成する手法を提案した．Achatz らが対象としたのは分割統治型のパワーリスト関数であり，その他のパワーリスト関数についてアルゴリズムを生成できない．我々は相互結合網をハイパキューブに制限することで Achatz らの手法を拡張した同種分解可能，交差分解可能と呼ぶ二つのパワーリスト関数のクラスを提案する．そして，それらの関数から時間計算量が $O(n)$ である $n$ 次元ハイパキューブアルゴリズムが生成可能であることを示す．

**キーワード**　パワーリスト，ハイパキューブ，並列アルゴリズム，プログラム導出

## 1. まえがき

パワーリストは並列アルゴリズム等の並列再帰構造を形式的かつ簡潔に記述するために，Misra [8] によって提案されたデータ構造である．パワーリストは $2^n$ 個の要素をもつリストであり，並列再帰構造をパワーリスト上の関数として記述する．パワーリストは $2^n$ 個の要素をもつため，相互結合網としての $n$ 次元ハイパキューブと親和性が高い．そのため，パワーリストによるハイパキューブアルゴリズムの仕様記述に関する研究が数多くある [7], [8], [10], [12].

パワーリストの代数的な性質は等式論理による関数の等価変換及び等価性判定を可能とするため，定理証明器を用いた自動検証研究 [4], [5] がある．また，パワーリストが対象とする並列再帰構造に着目し，相互結合網 [9]，加算回路 [3] の記述に関する研究がある．

本論文ではパワーリストによる仕様記述からアルゴリズムを生成することを目的とする．関連研究として Kornerup [6]，Achatz and Schulte [1], [2] がある．Kornerup はグレイコードを用いて，パワーリスト及びパワーリストに対する基本演算をハイパキューブ上に写像している．しかし，写像された基本演算からハイパキューブ上のアルゴリズムを生成する具体的な手法について言及していない．

Achatz and Schulte はパワーリスト関数を DC と呼ばれる高階関数によって記述し，それを相互結合網非依存の基本通信操作による記述に変換する手法を提案している [1]．高階関数 DC は分割統治型のパワーリスト関数を記述するためのテンプレートである．基本通信操作は超データ並列処理をモデルとし，実装可能な相互結合網としてメッシュ，アレー，ハイパキューブを考慮している．結果として，高階関数 DC による記述から，基本通信操作を実装可能な相互結合網におけるアルゴリズムが得られる．

パワーリストは 2 種類の基本構築子をもつが，DC によるパワーリスト関数の記述ではパワーリストの分解と構築のために同一の基本構築子を用いる必要がある．そのため，DC を用いた手法では同一基本構築子を含む分割統治型の関数のみを記述の対象としている．その後 Achatz らは異なる基本構築子を含む関数を記述するために，文献 [2] において新たな手法を提案している．具体的には inversion と呼ばれる操作を導入することで，異なる基本構築子によって引き起こされる再帰呼出しごとのデータ再配置を一括処理している．しかし，DC ではトップダウン，ボトムアップの処理が混在する関数を記述できるのに対し，新たな手法で

[†] 岩手大学大学院工学研究科，盛岡市
Graduate School of Science and Engineering, Iwate University, Morioka-shi, 020–8551 Japan
[††] 岩手大学工学部情報システム工学科，盛岡市
Department of Computer and Information Sciences, Faculty of Engineering, Iwate University, Morioka-shi, 020–8551 Japan

は関数をトップダウン関数あるいはボトムアップ関数のどちらか一方のテンプレートで記述する必要がある．もとの関数からテンプレートへの変形は非形式的に行われており，常に成功するとは限らない．

Achatz らの手法の利点は，相互結合網非依存のアルゴリズムが導出できることである．その反面，相互結合網非依存性と高速化のために記述の対象外としているパワーリスト関数が存在する．

本論文において，アルゴリズム生成の対象となるパワーリスト関数のクラス拡張のために，相互結合網をハイパキューブに制限することで Achatz らの手法を拡張する．我々は記述対象として（1）分割統治型/一部の非分割統治型，（2）同一の基本構築子/異なる基本構築子，（3）トップダウン，ボトムアップ処理が混在する関数を考えている．条件（2），（3）は Achatz らの 2 種類の手法が対象とする関数を組み合わせたものを表す．更に条件（1）により，非分割統治型の一部の関数についても新たに記述の対象とする．上記の関数は提案する高階関数 HDF，CDF によって記述される．HDF，CDF は高階関数 DC を非分割統治型及び異なる基本構築子についても記述できるように拡張したものである．高階関数 HDF，CDF で記述された関数を分解可能関数と呼ぶ．また，恒等関数と inversion のみを基本関数として，HDF，CDF に対するハイパキューブアルゴリズムを与える．結果として，分解可能関数から時間計算量が $O(n)$ である $n$ 次元ハイパキューブアルゴリズムが生成可能なことを示す．

**2.** ではパワーリストによるハイパキューブアルゴリズムの仕様記述及び高階関数 DC について述べる．**3.** において提案する新たな高階関数 HDF，CDF について述べ，**4.** でハイパキューブアルゴリズムの生成手法について述べる．**5.** では高階関数 HDF，CDF による記述例と生成されるアルゴリズムの例について述べる．

## 2. パワーリストによる仕様記述

### 2.1 パワーリスト

パワーリストは長さ（要素数）が $2^n$（$n \geq 0$）のリストであり，$\langle x_0\ x_1 \ldots\ x_{2^n-1}\rangle$ のように表される．長さが 1 のパワーリスト $\langle x\rangle$ をシングルトンと呼ぶ．パワーリストは再帰構造をもち，同じ長さのパワーリスト 2 個からより大きなパワーリストを構築できる．パワーリストの基本構築子として | （tie）と $\bowtie$（zip）がある．基本構築子についての公理を与えることによりパワーリスト及び基本構築子を代数的に定義できるが [8]，ここではリスト操作としての基本構築子の意味を説明して定義とする．$p$，$q$ を同じ長さのパワーリストとしたとき，$p \mid q$ は $p$ と $q$ を連結したパワーリストである．例えば $\langle 0\rangle \mid \langle 1\rangle = \langle 0\ 1\rangle$ であり，$\langle 0\ 1\ 2\ 3\rangle \mid \langle 4\ 5\ 6\ 7\rangle = \langle 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\rangle$ である．$p \bowtie q$ は $p$ と $q$ の要素を先頭から交互に並べたパワーリストである．例えば $\langle 0\rangle \bowtie \langle 1\rangle = \langle 0\ 1\rangle$ であり，$\langle 0\ 1\ 2\ 3\rangle \bowtie \langle 4\ 5\ 6\ 7\rangle = \langle 0\ 4\ 1\ 5\ 2\ 6\ 3\ 7\rangle$ である．シングルトンに対する $\bowtie$ と | の結果は同じであり，長さが 2 以上のパワーリストについては異なるリストとなる．

### 2.2 仕様記述

並列アルゴリズムの仕様記述をパワーリスト上の関数によって定義する．例として，パワーリストの要素の並びを逆順にする関数 $rev$ の定義を与える [8]．

$$\begin{aligned} &rev\ \langle x\rangle = \langle x\rangle \\ &rev\ (p \mid q) = (rev\ q) \mid (rev\ p) \end{aligned} \tag{1}$$

関数 $rev$ は基底の場合，シングルトンに対する恒等関数と等価である．再帰の場合，基本構築子が等号の両辺に現れているが，左辺のものを分解子，右辺のものを構築子と呼び区別する．引数は分解子 | により，長さが半分のパワーリスト $p$ と $q$ に分けられる．$p$ と $q$ は構築子 | をまたいで交換され，それぞれ再帰した結果が再び構築子で結合される．分解子で分けられたあとの再帰呼出しは互いに影響を及ぼさないので，並列に処理できる．

関数 $rev$ は構築子と分解子が | で一致している．そのような関数を整合な関数と呼ぶことにする．また，$p$ と $q$ に対する関数呼出しとして自分自身の再帰呼出しのみを含んでいる．そのような関数を分割統治型の関数と呼び，$p$ と $q$ の少なくとも一方について自分自身を呼び出していないものを非分割統治型の関数と呼ぶ．次の関数 $ls$ [8] は再帰呼出しがなく分解子が |，構築子が $\bowtie$ であるので非分割統治型の不整合な関数である．

$$\begin{aligned} &ls\ \langle x\rangle = \langle x\rangle \\ &ls\ (p \mid q) = p \bowtie q \end{aligned} \tag{2}$$

図 1 に $ls$ の適用例を示す．$ls$ は引数のパワーリスト要素の 2 進表示インデックスを左に回転する関数である．例えば，$(100)_2$ 番目の要素が $(001)_2$ 番目に，$(110)_2$ 番目の要素が $(101)_2$ に移動する．ここで $(xxx)_2$ は整数の 2 進表示を表す．

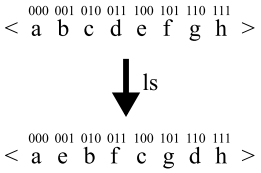



図 1 $ls$ 操 作
Fig. 1 $Ls$ operation.

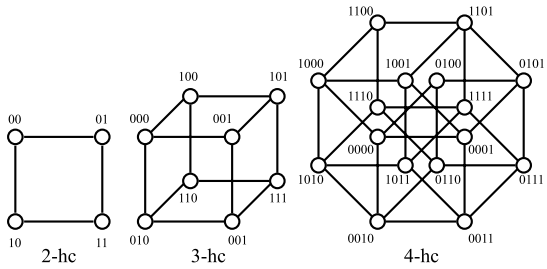



図 2 ハイパキューブ
Fig. 2 Hypercubes.

パワーリスト関数の中には，スカラ関数と呼ばれる特殊な関数が存在する．スカラ関数は要素の再配置を行わず，要素ごとに $f$ を適用した結果のパワーリストを返す．1 引数のスカラ関数 $f$ は以下のように定義される．

$$f\ \langle x\rangle = \langle f\ x\rangle$$
$$f\ (p \parallel q) = (f\ p) \parallel (f\ q)$$

ここで $\parallel$ は基本構築子 | または $\bowtie$ を表し，その種類によらず $f$ が定義可能であることを示している．

1 引数のスカラ関数と同様に 2 引数のスカラ関数も次のように定義され，これは $n$ 引数に一般化される．

$$f\ \langle x\rangle\ \langle y\rangle = \langle f\ x\ y\rangle$$
$$f\ (p \parallel q)\ (u \parallel v) = (f\ p\ u) \parallel (f\ q\ v)$$

### 2.3 ハイパキューブアルゴリズム

$n$ 次元ハイパキューブ（$n$-HC）は $2^n$ 個の頂点をもつグラフである（図 2）．頂点 $i$（$0 \le i \le 2^n - 1$）は $n$ ビットのラベルをもち，二つの頂点 $i$ と $j$ はそれらのラベルが $d$（$0 \le d \le n-1$）番目のビット位置だけで異なるとき，かつそのときに限り，辺をもつ．このとき $i$ について，隣接する頂点 $j$ を $i$ の $d$ 次元方向の頂点，あるいは単に $d$ 次元方向という．

グラフの頂点と辺をそれぞれプロセッサ，リンクとして考えると，ハイパキューブを並列システムとみなすことができる．以降では頂点をノードと呼ぶ．また，パワーリストによるアルゴリズムの仕様記述において，$n$-HC のノード $i$ がデータ $x_i$ をもつことをパワーリスト $\langle x_0\ x_1\ \ldots\ x_{2^n-1}\rangle$ によって表す．

並列アルゴリズムを考える場合，同期モデル，通信モデルを定める必要がある．本論文において，同期モデルはすべてのノードが共有した時刻に従って処理を行う同期方式とする．時刻で分けられる処理の単位をラウンドと呼ぶ．通信モデルについては各ラウンドでノードが通信する相手をたかだか一つに制限するシングルポート方式とする．また，一度の通信で送受信可能，つまりデータの交換が可能であるとする．

### 2.4 高階関数 DC

Achatz らは超データ並列アルゴリズムを導出するために，分割統治法による関数を高階関数 DC によって記述した [1]．DC による手法は以下の手順で並列アルゴリズムを生成する．

（1） パワーリスト関数 $f$ を DC で記述
（2） $f$ をトップダウン，ボトムアップ関数に分解
（3） 上記の分解された関数を基本通信操作による記述に変換

結果として，基本通信操作を実装可能であるアレー，メッシュ，ハイパキューブ上の並列アルゴリズムを生成できる．

上記手順のうち，本論文に関連する（1），（2）について詳しく説明する．まず，高階関数 DC の定義を与え，DC で記述できるパワーリスト関数を説明する．

DC は引数として $q$，$t$，$g$，$h$，$k$，$j$ をとり，関数を返す高階関数であり，次のように定義される．

$$f = \mathrm{DC}\ \ q\ t\ g\ h\ k\ j$$
$$f\ x = t\ x \qquad \textbf{if} \quad q = \#x$$
$$f\ (x\ \mid\ y) = (k\ v\ w)\ \mid\ (j\ v\ w) \quad \textbf{otherwise}$$
$$\textbf{where}\ (v,\ w) = (f\ (g\ x\ y),\ f\ (h\ x\ y))$$

ここで整数 $q$ は 2 のべき，$t$ は自明な関数と呼ばれるスカラ関数であり，$g$，$h$ は前調節関数，$k$，$j$ は後調節関数と呼ばれるスカラ関数である．$\#$ はパワーリストの長さを返す．自明な関数，前調節関数，後調節関数がスカラ関数であるのは，$f$ を実現する並列アルゴリズムを考えた場合に通信を必要としないためである．すなわち，これらの関数の適用が 1 ラウンド中で実行できることを保証する条件である．

DC は，関数 $f$ が分割統治法により処理できることを示している．再帰関数 $f$ は引数 $x$ の長さ $\#x$ が $q$

と等しいときに基底となり，自明な関数 $t$ を適用した結果を返す．再帰の場合，引数を $|$ で分解した $x$，$y$ に対して前調節関数 $g$，$h$ を適用し，その結果である $g\ x\ y$，$h\ x\ y$ についてそれぞれ $f$ で再帰する．その後，再帰呼出しの結果 $v$，$w$ に対して後調節関数 $k$，$j$ が適用され，それらを $|$ で結合したものが返される．DC は分解子と構築子が整合していればよいので $|$ の代わりに $\bowtie$ を用いた定義も考えられるが，Achatz らは相互結合網非依存を達成するために $|$ のみを基本構築子としている [1]．

DC において $k\ v\ w = v$，$j\ v\ w = w$ であれば，$f$ の再帰定義は $f\ (x\ |\ y) = f\ (g\ x\ y) \mid f\ (h\ x\ y)$ となり，再帰的な計算結果を結合するだけの単純なトップダウンアルゴリズムが生成できる．逆に $g\ x\ y = x$，$h\ x\ y = y$ であればボトムアップに $f$ を計算でき，その並列アルゴリズムも容易に生成できる．

このように DC をトップダウン/ボトムアップに制限した高階関数 $\mathrm{DC}^{\downarrow}$，$\mathrm{DC}^{\uparrow}$ を次のように定義する．

$$\mathrm{DC}^{\downarrow}\ q\ t\ g\ h = \mathrm{DC}\ \ q\ t\ g\ h\ \pi_0\ \pi_1$$
$$\mathrm{DC}^{\uparrow}\ q\ t\ k\ j = \mathrm{DC}\ \ q\ t\ \pi_0\ \pi_1\ k\ j$$

ここで，$\pi_i$ は射影関数であり，$\pi_0\ x\ y = x$，$\pi_1\ x\ y = y$ である．$\mathrm{DC}^{\downarrow}$，$\mathrm{DC}^{\uparrow}$ によって記述される関数をそれぞれトップダウン関数，ボトムアップ関数と呼ぶ．

Achatz らは DC で記述された関数について次のことを示している．

> スカラ関数 $t$，$t^{\uparrow}$，$t^{\downarrow}$ について $t = t^{\uparrow} \circ t^{\downarrow}$ とし，$f = \mathrm{DC}\ \ q\ t\ g\ h\ k\ j$，$f^{\downarrow} = \mathrm{DC}^{\downarrow}\ q\ t^{\downarrow}\ g\ h$，$f^{\uparrow} = \mathrm{DC}^{\uparrow}\ q\ t^{\uparrow}\ k\ j$ としたとき，$f = f^{\uparrow} \circ f^{\downarrow}$ が成り立つ．ここで $\circ$ は関数合成を表す．

これは DC で記述された関数 $f$ が，トップダウン関数 $f^{\downarrow}$ とボトムアップ関数 $f^{\uparrow}$ に分解できることを示しており，$f$ の並列アルゴリズムを考える場合に $f^{\downarrow}$，$f^{\uparrow}$ のアルゴリズムを考えればよいことになる．つまり DC で記述できる関数ならば並列アルゴリズムを生成できる．

DC による関数の記述例として，次のように定義された $rev'$ を考える．ここで $id$ は恒等関数を表す．

$$rev' = \mathrm{DC}\ 1\ id\ \pi_1\ \pi_0\ \pi_0\ \pi_1$$

このとき以下が成立し，$rev$ の定義式 (1) と一致する．

$$rev'\ \langle x\rangle = id\ \langle x\rangle = \langle x\rangle$$
$$rev'\ (x \mid y)$$
$$= (\pi_0\ v\ w) \mid (\pi_1\ v\ w)$$
$$\quad \textbf{where}\ (v,\ w) = (rev'\ (\pi_1\ x\ y),\ rev'\ (\pi_0\ x\ y))$$
$$= v \mid w$$
$$\quad \textbf{where}\ (v,\ w) = (rev'\ y,\ rev'\ x)$$
$$= (rev'\ y) \mid (rev'\ x)$$

高階関数 DC によるパワーリスト関数の記述には制限が存在する．例として，次の関数 $rr$ [8] を考える．

$$\begin{aligned} rr\ \langle x\rangle &= \langle x\rangle \\ rr\ (p \bowtie q) &= (rr\ q) \bowtie p \end{aligned} \tag{3}$$

関数 $rr$ は引数のパワーリストを要素一つ分だけ右に回転する関数であり，$rr\ \langle 0\ 1\ 2\ 3\rangle = \langle 3\ 0\ 1\ 2\rangle$ である．$rr$ は基本構築子 $\bowtie$ を用いているので DC では記述できないが，DC の定義を $\bowtie$ に拡張したとしても $rr$ は構築子の右の部分式において再帰呼出しがない．そのため，分割統治型の関数を対象とする DC では記述できない．また，DC は基本構築子 $|$ で整合しているので，式 (2) で定義される不整合な関数 $ls$ についても記述することができない．

DC で記述可能な関数は（1）分割統治型，（2）整合，（3）トップダウン，ボトムアップ処理が混在した関数である．関数 $rr$，$ls$ は条件 （1），（2）を満たさないので，DC では記述できない．

Achatz らは文献 [2] において不整合な関数を記述するために新たな高階関数を提案しており，（1）分割統治型，（2）整合/不整合，（3）トップダウン，ボトムアップ処理が混在しない関数を対象としている．Achatz らは不整合な関数を記述するために，基本操作として inversion を導入し，もとの関数から $\bowtie$ を除去するための変換規則を提案している．変換規則はトップダウン処理とボトムアップ処理の場合で異なり，$\bowtie$ が分解子，構築子，またはその両方に現れる場合によっても異なっているため，複数存在する．それらの規則により，関数は基本構築子 $|$ のみを含む整合な関数へと変換され，アルゴリズムの生成が可能となる．非分割統治型の関数については，DC と同様に記述できない．また，不整合な関数について記述が可能になった反面，与えられた関数をトップダウン関数若しくはボトムアップ関数の形式でしか記述できない．したがって，トップダウン，ボトムアップ処理の混在した関数をトップダウン関数若しくはボトムアップ関数に変形する必要があるが，その方法については非形式的である．

## 3. 分解可能関数

前章で述べた DC についての制限を緩め，より多くのパワーリスト関数からのハイパキューブアルゴリズムの生成を考える．我々は記述の対象として（1）分割統治型/一部の非分割統治型，（2）整合/不整合，（3）トップダウン，ボトムアップ処理が混在した関数を考えている．

アルゴリズム生成全体の手順は次のようになる．

（1） パワーリスト関数 $f$ を HDF か CDF で記述（HDF，CDF は DC を拡張した高階関数で，後述する）

（2） $f$ をトップダウン，ボトムアップ関数に分解

（3） 分解した関数それぞれのハイパキューブアルゴリズムを生成

この章では（1），（2）に対応する高階関数 HDF，CDF の定義とそれらで記述される関数の分解について説明する．また（3）については，**4.** で説明する．

### 3.1 同種分解可能関数

我々の提案手法において，恒等関数 $id$ と inversion 関数 $inv$ を関数定義のための基本関数とする．$inv$ はパワーリスト要素の 2 進表示インデックスを逆順にする関数である．例えば，$(011)_2$ 番目の要素が $(110)_2$ 番目に移動し，$(101)_2$ 番目の要素は移動しない（図 3）．

［定義 1］（基本関数） 基本関数 $id$，$inv$ はそれぞれ以下のように定義される関数である．ここで $\|$ は $|$ または $\bowtie$ であり，$\bar{\|}$ は $\|$ と異なる基本構築子である．

$$id\ \langle x\rangle = \langle x\rangle$$
$$id\ (p \,\|\, q) = (id\ p) \,\|\, (id\ q)$$
$$inv\ \langle x\rangle = \langle x\rangle$$
$$inv\ (p \,\|\, q) = (inv\ p)\,\bar{\|}\,(inv\ q)$$

$id$ は関数呼出しがないことを明示的に記述するために用いられる．$inv$ はのちに HDF と CDF を関連づける際に大きな役割を果たす．

基本関数 $id$，$inv$ 及び高階関数 DC を拡張した HDF を用いてパワーリスト上の整合な関数のクラスを再帰的に定義する．

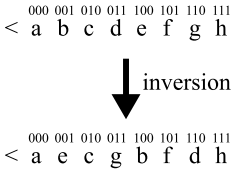



図 3 inversion 操作
Fig. 3 Inversion operation.

［定義 2］（同種分解可能関数） 基本関数 $id$ と $inv$ を 0 次同種分解可能関数とする．$m+1$ 次同種分解可能関数の集合 $H_{m+1}$ は高階関数 HDF により以下のように再帰的に定義される関数 $f$ の集合である．ここで $t,\ g,\ h,\ k,\ j$ はスカラ関数，$f_0,\ f_1 \in \bigcup_{i=0}^{m} H_i \cup \{f\}$ とし，$f_0$，$f_1$ の少なくとも一方は $m$ 次同種分解可能関数であるとする（$f_0 = f_1 = f$ のときは，下記の定義より $f$ は 1 次同種分解可能関数となる）．

$$f = \text{HDF}\ (\|)\ t\ f_0\ f_1\ g\ h\ k\ j$$
$$f\ \langle x\rangle = t\ \langle x\rangle$$
$$f\ (p \,\|\, q) = (k\ v\ w) \,\|\, (j\ v\ w)$$
$$\quad \textbf{where}\ (v,\ w) = (f_0\ (g\ p\ q),\ f_1\ (h\ p\ q))$$

高階関数 HDF は，DC に関して基本構築子を $\bowtie$ についても扱えるようにし，更に，定義中で再帰以外の関数 $f_0$，$f_1$ を呼び出せるように拡張している．したがって，DC $1\ t\ g\ h\ k\ j$ は HDF を用いて HDF $(|)\ t\ f\ f\ g\ h\ k\ j$ と記述できる．

HDF で記述できる関数は（1）分割統治型/一部の非分割統治型，（2）整合，（3）トップダウン，ボトムアップ処理が混在した関数である．

前章で述べた DC で記述できない非分割統治型の関数 $rr$ は 1 次の同種分解可能関数であり，HDF で記述できる．すなわち，$rr' = \text{HDF}\ (\bowtie)\ id\ rr'\ id\ \pi_1\ \pi_0\ \pi_0\ \pi_1$ として，HDF の定義で展開すると $rr$ の定義 (3) と一致する．展開を以下に示す．

$$rr'\ \langle x\rangle = id\ \langle x\rangle = \langle x\rangle$$
$$rr'\ (p \bowtie q)$$
$$= (\pi_0\ v\ w) \bowtie (\pi_1\ v\ w)$$
$$\quad \textbf{where}\ (v,\ w) = (rr'\ (\pi_1\ p\ q),\ id\ (\pi_0\ p\ q))$$
$$= v \bowtie w$$
$$\quad \textbf{where}\ (v,\ w) = (rr'\ q,\ id\ p)$$
$$= (rr'\ q) \bowtie p$$

ハイパキューブアルゴリズムを考えた場合，DC と同様に HDF もトップダウン関数とボトムアップ関数に分解できればその生成は容易になる．HDF が分解できることを示すために，トップダウン関数，ボトムアップ関数を記述する高階関数をそれぞれ $\text{HDF}^{\downarrow}$，$\text{HDF}^{\uparrow}$ として定義する．

[定義 3]（HDF トップダウン関数） 基本関数 $id$ と $inv$ はトップダウン関数であると定義する．$f_0$，$f_1$ がトップダウン関数であるとき，高階関数 HDF によってトップダウン高階関数 $\mathrm{HDF}^{\downarrow}$ を以下のように定義する．

$$\begin{aligned}&\mathrm{HDF}^{\downarrow}\ (\|)\ t\ f_0\ f_1\ g\ h\\&\quad=\mathrm{HDF}\ (\|)\ t\ f_0\ f_1\ g\ h\ \pi_0\ \pi_1\end{aligned}$$

[定義 4]（HDF ボトムアップ関数） 基本関数 $id$ と $inv$ はボトムアップ関数であると定義する．$f_0$，$f_1$ がボトムアップ関数であるとき，高階関数 HDF によってボトムアップ高階関数 $\mathrm{HDF}^{\uparrow}$ を以下のように定義する．

$$\begin{aligned}&\mathrm{HDF}^{\uparrow}\ (\|)\ t\ f_0\ f_1\ k\ j\\&\quad=\mathrm{HDF}\ (\|)\ t\ f_0\ f_1\ \pi_0\ \pi_1\ k\ j\end{aligned}$$

基本関数 $id$ と $inv$ はトップダウン関数かつボトムアップ関数であることから，次の補題が成り立つ．

[補題 5]（0 次同種分解可能関数の分解） $id = id \circ id$，$inv = id \circ inv$ であることから，$id^{\downarrow} = id$，$id^{\uparrow} = id$，$inv^{\downarrow} = inv$，$inv^{\uparrow} = id$ としたとき $id = id^{\uparrow} \circ id^{\downarrow}$，$inv = id^{\uparrow} \circ inv^{\downarrow}$ が成り立つ．すなわち，基本関数はトップダウン関数とボトムアップ関数に分解できる．

HDF で定義された関数が分解可能であることを示す．

[定理 6]（HDF の分解） 同種分解可能関数 $f$ について，$f = f^{\uparrow} \circ f^{\downarrow}$ である HDF トップダウン関数 $f^{\downarrow}$ と HDF ボトムアップ関数 $f^{\uparrow}$ が存在する．

（証明） 次数 $m$ に関する帰納法で証明する．基底 $m = 0$ の場合，補題 5 より定理は成り立つ．帰納の場合，$m$ 以下の次数で定理が成り立つと仮定する．関数 $f \in H_{m+1}$ が $f = \mathrm{HDF}\ (\|)\ t\ f_0\ f_1\ g\ h\ k\ j$ で定義されるとき，$f^{\downarrow} = \mathrm{HDF}^{\downarrow}\ (\|)\ t\ f_0^{\downarrow}\ f_1^{\downarrow}\ g\ h$，$f^{\uparrow} = \mathrm{HDF}^{\uparrow}\ (\|)\ id\ f_0^{\uparrow}\ f_1^{\uparrow}\ k\ j$ として，パワーリストの長さに関する帰納法で示す．ここで $f^{\downarrow}$，$f^{\uparrow}$ の自明な関数 $t^{\downarrow}$，$t^{\uparrow}$ は $t = t^{\uparrow} \circ t^{\downarrow}$ を満足しなければならないので，$t^{\downarrow} = t$，$t^{\uparrow} = id$ としている．

（1） 基底の場合，$f^{\downarrow}$，$f^{\uparrow}$ の定義よりシングルトンに関して $f^{\downarrow}\ \langle x\rangle = t\ \langle x\rangle$，$f^{\uparrow}\ \langle x\rangle = id\ \langle x\rangle$ であり，$f^{\uparrow} \circ f^{\downarrow}\langle x\rangle = id\ (t\ \langle x\rangle) = t\ \langle x\rangle = f\ \langle x\rangle$ より成り立つ．

（2） 帰納の場合，長さ $2^l$ の任意のパワーリスト $p$ について $f^{\uparrow} \circ f^{\downarrow}\ p = f\ p$ であると仮定して，$\#(p_0 \parallel p_1) = 2^{l+1}$ の場合を考える．

$$\begin{aligned}&f^{\uparrow} \circ f^{\downarrow}(p_0 \parallel p_1)\\&=\{f^{\downarrow}\ \text{の定義}\}\\&f^{\uparrow}(f_0^{\downarrow}(g\ p_0\ p_1) \parallel f_1^{\downarrow}(h\ p_0\ p_1))\\&=\{f^{\uparrow}\ \text{の定義}\}\\&(k\ v\ w) \parallel (j\ v\ w)\\&\quad\textbf{where}\ (v,\ w) = (f_0^{\uparrow}(f_0^{\downarrow}(g\ p_0\ p_1)),\\&\qquad\qquad f_1^{\uparrow}(f_1^{\downarrow}(h\ p_0\ p_1)))\end{aligned}$$

このとき $g\ p_0\ p_1 = p_0'$，$h\ p_0\ p_1 = p_1'$ として，$i = 0, 1$ について $f_i^{\uparrow} \circ f_i^{\downarrow}\ p_i' = f_i\ p_i'$ が成り立てば，$f\ (p \parallel q) = f^{\uparrow} \circ f^{\downarrow}\ (p \parallel q)$ が成り立つ．$f_i$ の場合分けにより示す．

（a） $f_i \in \bigcup_{i=0}^{m} H_i$ のとき，次数 $m$ に関する帰納法の仮定より $f_i^{\uparrow} \circ f_i^{\downarrow} = f_i$ なので $f_i^{\uparrow} \circ f_i^{\downarrow}\ p_i = f_i\ p_i$.

（b） $f_i = f$ のとき，$\#p_i = 2^l$ なのでパワーリストの長さに関する帰納法の仮定より $f^{\uparrow} \circ f^{\downarrow}\ p_i = f\ p_i$ であるので $f_i^{\uparrow} \circ f_i^{\downarrow}\ p_i = f_i\ p_i$ が成り立つ．

したがって，定理が成り立つ． □

### 3.2 交差分解可能関数

前節に続き，基本関数 $id$，$inv$ 及び高階関数 DC を拡張した関数 CDF からパワーリスト上の不整合な関数のクラスを再帰的に定義する．

[定義 7]（交差分解可能関数） 関数 $id$ と $inv$ を 0 次交差分解可能関数とする．$m + 1$ 次交差分解可能関数の集合 $C_{m+1}$ は高階関数 CDF により以下のように再帰的に定義される関数 $f$ の集合である．ここで $\bar{\|}$ は $\|$ と異なる基本構築子，$t$，$g$，$h$，$k$，$j$ はスカラ関数，$f_0$，$f_1 \in \bigcup_{i=0}^{m} C_i \cup \{f\}$ とし，$f_0$，$f_1$ の少なくとも一方は $m$ 次交差分解可能関数であるとする（$f_0 = f_1 = f$ のときは，下記の定義より $f$ は 1 次交差分解可能関数となる）．

$$\begin{aligned}&f = \mathrm{CDF}\ (\|)\ t\ f_0\ f_1\ g\ h\ k\ j\\&f\ \langle x\rangle = t\ \langle x\rangle\\&f\ (p \parallel q) = (k\ v\ w)\bar{\|}(j\ v\ w)\\&\quad\textbf{where}\ (v,\ w) = (f_0\ (g\ p\ q),\ f_1\ (h\ p\ q))\end{aligned}$$

CDF で記述できるのは（1）分割統治型/一部の非分割統治型，（2）不整合，（3）トップダウン，ボトムアップ処理が混在した関数である．

前節で述べた DC で記述できない不整合な関数 $ls$

は 1 次の交差分解可能関数であり，CDF で記述できる．すなわち，$ls' = \text{CDF}\ (|)\ id\ id\ id\ \pi_0\ \pi_1\ \pi_0\ \pi_1$ として，CDF の定義で展開すると $ls$ の定義 (2) と一致する．

高階関数 CDF によって定義された関数が分解可能であることを示すために，CDF トップダウン関数を記述する高階関数を定義する．CDF で記述された関数の分解において，そのボトムアップ関数を HDF で定義するため，CDF ボトムアップ関数を記述する高階関数は不要である．

［定義 8］（CDF トップダウン関数） 基本関数 $id$ と $inv$ はトップダウン関数であると定義する．$f_0$，$f_1$ がトップダウン関数であるとき，高階関数 CDF によってトップダウン関数 $\text{CDF}^{\downarrow}$ を以下のように定義する．

$$\begin{aligned}&\text{CDF}^{\downarrow}\ (\|)\ t\ f_0\ f_1\ g\ h\\ &\quad = \text{CDF}\ (\|)\ t\ f_0\ f_1\ g\ h\ \pi_0\ \pi_1\end{aligned}$$

HDF 同様，CDF で定義された関数についても分解可能である．

［定理 9］（CDF の分解） 交差分解可能関数 $f$ について，$f = f^{\uparrow} \circ f^{\downarrow}$ である CDF トップダウン関数 $f^{\downarrow}$ と HDF ボトムアップ関数 $f^{\uparrow}$ が存在する．

（証明） $f^{\downarrow} = \text{CDF}^{\downarrow}\ (\|)\ t\ f_0^{\downarrow}\ f_1^{\downarrow}\ g\ h$，$f^{\uparrow} = \text{HDF}^{\uparrow}\ (\bar{\|})\ id\ f_0^{\uparrow}\ f_1^{\uparrow}\ k\ j$ として，定理 6 と同様に証明できる． □

ハイパキューブアルゴリズムを生成するには，不整合な関数を整合な関数に変換する必要がある．そのために Achatz らと同様に $inv$ を用いる．ただし **2.4** で述べたように Achatz らが $\bowtie$ を除去するために複数の変換規則を必要としたのに対して，我々は分割統治型/非分割統治型の不整合なトップダウン関数の構築子に対する単一の変換規則のみを用いる．

［定理 10］（交差同種変換） CDF トップダウン関数 $f^{\downarrow}$ が与えられたとき，$inv \circ f^{\downarrow}$ は HDF トップダウン関数である．

（証明） $f^{\downarrow} = \text{CDF}^{\downarrow}\ (\|)\ t\ f_0\ f_1\ g\ h$，$f^{\downarrow} \in C_{m+1}$ として，定理 6 と同様に次数 $m$ に関する帰納法で証明する．基底 $m = 0$ の場合，$f^{\downarrow} \in \{id,\ inv\}$ より，

$$inv \circ id = inv \in H_0, \quad inv \circ inv = id \in H_0$$

帰納の場合，$m$ 以下の次数で成り立つとして，$m+1$ の場合をパワーリストの長さに関する帰納法により $f_H = inv \circ f^{\downarrow}$ として証明する．基底（シングルトン）の場合，

$$f_H\ \langle x \rangle = inv \circ f^{\downarrow} \langle x \rangle = inv\ (t\ \langle x \rangle) = t\ \langle x \rangle$$

帰納の場合，長さ $2^l$ の任意のパワーリスト $p$ について $inv \circ f^{\downarrow}\ p = f_H\ p$ であると仮定して，$\#(p_0 \parallel p_1) = 2^{l+1}$ の場合を考える．

$$\begin{aligned}&f_H\ (p_0 \parallel p_1)\\ &\quad = inv \circ f^{\downarrow}(p_0 \parallel p_1)\\ &\quad = inv\ (f_0\ (g\ p_0\ p_1)\ \bar{\|}\ f_1\ (h\ p_0\ p_1))\\ &\quad = inv\ (f_0\ (g\ p_0\ p_1)) \parallel inv\ (f_1\ (h\ p_1\ p_1))\end{aligned}$$

このとき $g\ p_0\ p_1 = p_0'$，$h\ p_0\ p_1 = p_1'$ として，$inv \circ f_i\ p_i'$ $(i = 0, 1)$ について $inv \circ f_i \in \bigcup_{i=0}^{m} H_i \cup \{f_H\}$ が成り立てば，$f_H\ (p \parallel q) \in H_{m+1}$ が成り立つ．$f_i \in \bigcup_{i=0}^{m} C_i$ のとき，次数 $m$ に関する帰納法の仮定より $inv \circ f_i \in \bigcup_{i=1}^{m} H_i$ である．$f_i = f^{\downarrow}$ のときは $inv \circ f_i\ p_i' = inv \circ f^{\downarrow}\ p_i'$ であり，パワーリストの長さに関する帰納法の仮定より $inv \circ f_i\ p_i' = f_H\ p_i'$ である．したがって，$inv \circ f_i \in \bigcup_{i=0}^{m} H_i \cup \{f_H\}$ が成り立つ． □

交差同種変換は，$\text{CDF}^{\downarrow}$ が $inv$ と $\text{HDF}^{\downarrow}$ の合成で表現できることを示している．このことから $inv$，$\text{HDF}^{\downarrow}$，$\text{HDF}^{\uparrow}$ のアルゴリズムが存在すれば，CDF のアルゴリズムが生成できる．

## 4. アルゴリズム生成

HDF，CDF で記述された関数は $\text{HDF}^{\downarrow}$，$\text{HDF}^{\uparrow}$ で記述された関数及び $inv$ に分解できる．HDF，CDF による記述からのアルゴリズムの生成のために，分解されたそれらのアルゴリズムがハイパキューブ上で実現できることを示す．

### 4.1 inversion アルゴリズム

$n$ 次元ハイパキューブ $n$-HC 上の inversion アルゴリズム $\text{INV}(u,\ l)$ はノード番号における下位ビット位置 $l$ から上位ビット位置 $u$ までの連続した部分ビット列を対象とし，その部分ビット列を逆順にしたノードとデータを交換する．

inversion は BCP（Bit Permute Complement）と呼ばれる置換のクラスに含まれており，$n$-HC において BCP は時間計算量 $O(n)$ で実現できることが知られている [11]．そのアルゴリズムを用いることで，$n$-HC における $\text{INV}(n-1,\ 0)$ は $n$ ラウンド以下で実行できる．

### 4.2 同種分解可能な関数からのアルゴリズム生成

HDF で記述された関数 $f$ は，トップダウン関数と

ボトムアップ関数に分解できる．そのため HDF$^{\downarrow}$ 及び HDF$^{\uparrow}$ がハイパキューブ上で実現可能であれば，HDF で記述されたもとの関数 $f$ も実現可能である．

分解子 $\|$ が $|$ のときの HDF$^{\downarrow}$ の $n$-HC アルゴリズムを説明するために，$f^{\downarrow} = \text{HDF}^{\downarrow}\ (|)\ t\ f_0^{\downarrow}\ f_1^{\downarrow}\ g\ h$ である関数の定義式を示す．

$$f^{\downarrow}\langle x\rangle = t\ \langle x\rangle$$
$$f^{\downarrow}(p \mid q) = f_0^{\downarrow}\ (g\ p\ q) \mid f_1^{\downarrow}\ (h\ p\ q)$$

アルゴリズムは引数として HDF で定義された分解前の関数 $f$ を取り，自明な関数 $t$ や分解子 $\|$ 等のパラメータを参照可能であるとする．

開始ラウンド 0 において，ラベルの $n-1$ 番目のビットが 0 のノードはパワーリスト $p$ の要素を保持するので，前調節関数 $g$ の適用のために対応するパワーリスト $q$ の要素を必要とする．同様に $q$ の要素を保持するノード（ラベルの $n-1$ 番目のビットが 1）も $h$ の適用のために $p$ の要素を必要とする．そのために各ノードは $n-1$ 次元方向のノードとデータの交換を行い，受信データと自分自身のデータに対して前調節関数を適用する．

次のラウンドで $p$ と $q$ に対応するノードは独立した $n-1$ 次元の部分ハイパキューブ上で，それぞれ関数 $f_0$，$f_1$ の処理を再帰的に繰り返し実行する．各ラウンドでノード $i$ が処理する関数は，$i$ のラベルに従って $f$ の関数呼出しをたどることで決定できる．例えば，$f$ が呼び出す関数の参照を $f.f_i$ と記述すれば，ラベルが 011 であるノードが処理する関数は順に $f$，$f.f_0$，$(f.f_0).f_1$，$((f.f_0).f_1).f_1$ である．

以上をまとめたアルゴリズム HDF$^{\downarrow}$ を与える．アルゴリズムの各ラウンド $r$ において，ノード $i$ は以下の処理を繰り返す．

（1） 現在の処理が $id$，あるいは現在のラウンド $r$ が次元 $n$ と等しいならば，$t$ をデータに適用してアルゴリズム終了．

（2） 現在の処理が $inv$ ならば，INV$(n-r-1,\ 0)$ を呼び出して終了．

（3） $n-r-1$ 次元方向のノードとデータを交換．

（4） $i$ のラベルの $n-r-1$ 番目のビット $b(i)[n-r-1]$ が 0 ならば第 1 引数を自分のデータ（$x_i$），第 2 引数を受信データ（$tmp$）として $g$ を適用し，1 ならば引数の順序を逆にして $h$ を適用．

分解子が $|$ であるときの HDF$^{\downarrow}$ アルゴリズムのテンプレートを図 4 に示す．アルゴリズムの生成例として，パワーリスト要素の最大値を求める関数 $max$ のアルゴリズムを **5. 1** に与える．

```
1:  global variable: n, r
2:  local variable: i, x_i, tmp
3:  procedure HDF↓(f)
4:      if (f = id) ∨ (r = n) then
5:          f.t (x_i)
6:          exit
7:      else if f = inv then
8:          INV(n − r − 1, 0)
9:          exit
10:     end if
11:
12:     send_recv(x_i, tmp, n − r − 1)
13:
14:     if b(i)[n − r − 1] = 0 then
15:         x_i ← f.g (x_i, tmp)
16:         f ← f.f_0
17:     else
18:         x_i ← f.h (tmp, x_i)
19:         f ← f.f_1
20:     end if
21: end procedure
```

図 4 HDF$^{\downarrow}$ アルゴリズム
Fig. 4 HDF$^{\downarrow}$ algorithm.

分解子 $\|$ が $\bowtie$ の場合の HDF$^{\downarrow}$ アルゴリズムは，上記アルゴリズムでの INV の呼出し INV$(n-r-1,\ 0)$ と次元方向 $n-r-1$ をそれぞれ INV$(n-1,\ r)$ と $r$ で置き換えればよい．

次に分解子 $\|$ が $|$ のときの HDF$^{\uparrow}$ ハイパキューブアルゴリズムを説明する．説明のために $f^{\uparrow} = \text{HDF}^{\uparrow}\ (\|)\ id\ f_0^{\uparrow}\ f_1^{\uparrow}\ k\ j$ である関数の定義式を示す．

$$f^{\uparrow}\langle x\rangle = id\ \langle x\rangle$$
$$f^{\uparrow}(p \mid q) = (k\ v\ w) \mid (j\ v\ w)$$
$$\textbf{where}\ (v,\ w) = (f_0^{\uparrow}\ p,\ f_1^{\uparrow}\ q)$$

HDF$^{\uparrow}$ が HDF$^{\downarrow}$ と異なるのは関数呼出し $f_i$ とスカラ関数 $k$，$j$ の適用の順序である．HDF$^{\downarrow}$ がスカラ関数を適用したあとで関数呼出しを実行するのに対し，HDF$^{\uparrow}$ はそれを逆に行う．このことから，HDF$^{\uparrow}$ はまずシングルトンに $id$ を適用し，それをシングルトンに対する関数呼出しの結果として返す．それから $k$，$j$ を適用し，それらを構築子 $|$ で結合したものを更に上位の関数呼出しの結果として返す．

つまり HDF$^{\downarrow}$ が $n$-HC を独立な部分ハイパキューブに分割しながら処理を進めるのに対し，HDF$^{\uparrow}$ は独立な 0-HC から処理結果を結合しながら処理を進める．そのため HDF$^{\downarrow}$ が各ラウンドで $n-1$ 次元方向から 0 次元方向まで順に通信するのに対して，HDF$^{\uparrow}$ はその逆順に通信を行う．つまり，ラウンド $r$ でのノード $i$ のデータの交換を考えた場合，HDF$^{\downarrow}$ において $i$ は

$n-r-1$ 次元方向と通信を行うが，HDF$^{\uparrow}$ では $r$ 次元方向と通信を行う．

各ラウンドでノード $i$ が処理する関数についてもその順序は HDF$^{\downarrow}$ と逆になる．例えば，ラベルが 011 であるノードが処理する関数は順に $((f.f_0).f_1).f_1$，$(f.f_0).f_1$，$f.f_0$，$f$ である．

以上をまとめたアルゴリズム HDF$^{\uparrow}$ を与える．アルゴリズムの各ラウンド $r$ において，ノード $i$ は以下の処理を繰り返す．

（1） ラウンド $r$ が $n$ ならばアルゴリズム終了．

（2） 現在の処理が $inv$ ならば $f$ から $inv$ が呼び出されるまでの関数呼出しの数を $c$ とし，INV$(n-c-1,\ 0)$ を呼び出して処理の終了を待つ．

（3） $r$ 次元方向のノードとデータを交換する．

（4） $i$ の $r$ 番目のビットが 0 ならば第 1 引数を自分のデータ，第 2 引数を受信データとして $k$ を適用．そうでなければ引数の順序を逆にして $h$ を適用．

分解子 $\|$ が $\bowtie$ のときは，上記アルゴリズムでの INV の呼び出し INV$(n-c-1,\ 0)$ と次元方向 $r$ をそれぞれ INV$(n-1,\ c)$ と $n-r-1$ で置き換えればよい．

HDF$^{\downarrow}$，HDF$^{\uparrow}$ についての上記のアルゴリズムは $n$ ラウンドで実行できる．したがって定理 6 より，次の定理が成り立つ．

[定理 11]（HDF の実現） 同種分解可能関数 $f = f^{\uparrow} \circ f^{\downarrow}$ は HDF$^{\downarrow}$，HDF$^{\uparrow}$ を順に実行することにより $n$-HC 上で $O(n)$ ラウンドで実行可能である．

### 4.3 交差分解可能な関数からのアルゴリズム生成

交差分解可能関数 CDF について，定理 9，10 と HDF$^{\downarrow}$，HDF$^{\uparrow}$，$inv$ が $n$ ラウンドで実行できることより，次の定理が成り立つ．

[定理 12]（CDF の実現） 交差分解可能関数 $f$ は $n$-HC 上で $O(n)$ ラウンドで実行可能である．

（証明） 定理 9，10 より，$f_H = inv \circ f^{\downarrow} \in H_m$ としたとき，$f = f^{\uparrow} \circ \mathrm{inv} \circ f_H$ が成り立つ．したがって，HDF$^{\downarrow}$，INV，HDF$^{\uparrow}$ を順に実行することで，ハイパキューブ上で $f$ を実行できる． □

## 5. パワーリスト関数の記述例

提案した高階関数 HDF，CDF によるスカラ関数を含んだパワーリスト関数の記述例とアルゴリズムの生成例を示す．例の中には Achatz らの手法 [1], [2] では記述できない関数が含まれている．

### 5.1 HDF の記述例

スカラ関数を用いる整合な関数の例として，与えられたパワーリスト要素の最大値を求める関数 $max$ を考える．$max$ は非分割統治型の関数として定義される．ここで与える $max$ は最大値をパワーリストの先頭に再配置する関数である．

$$max\ \langle x\rangle = id\ \langle x\rangle$$
$$max(p \mid q) = max\ (ge\ p\ q) \mid id\ (le\ p\ q)$$

$ge$（$le$）はパワーリストの要素を比較して，大きい（小さい）ものを要素とするパワーリストを返すスカラ関数である．例えば $ge\ \langle 5\ 2\rangle\ \langle 3\ 6\rangle = \langle 5\ 6\rangle$，$le\ \langle 5\ 2\rangle\ \langle 3\ 6\rangle = \langle 3\ 2\rangle$ であり，$max\ \langle 4\ 1\ 3\ 7\rangle = \langle 7\ 4\ 3\ 1\rangle$ である．

$max$ は再帰呼出し以外に関数 $id$ を呼出している（すなわち，関数呼出しがない）ので非分割統治型の関数であり，Achatz らの手法では記述できない．Achatz らは最大値を求める関数を記述するために，$max'\ \langle 4\ 1\ 3\ 7\rangle = \langle 7\ 7\ 7\ 7\rangle$ のようにパワーリストのすべての要素を最大値で置き換えるように関数定義を変更している．

$max$ を HDF で記述すると次のようになる．

$$max = \mathrm{HDF}\ (|)\ id\ max\ id\ ge\ le\ \pi_0\ \pi_1$$

$max$ は後調節関数が $\pi_0, \pi_1$ なので，HDF トップダウン関数である．したがって，HDF$^{\downarrow}$ で記述できる．

$$max = \mathrm{HDF}^{\downarrow}\ (|)\ id\ max\ id\ ge\ le$$

$max$ は HDF トップダウン関数なので，**4.2** の手法によって図 5 に示すハイパキューブアルゴリズムが生成できる．大域変数 $n$，$r$ はハイパキューブの次元，現在のラウンドを表し，局所変数 $i$，$x_i$，$f$，$tmp$ はそれぞれ自ノード番号，パワーリストのデータ，現在の処理，受信用の一時変数を表す．また，現在の処理を示す変数 $f$ は初期状態で $f = max$ であるとする．簡単のために，不必要な $inv$ などの処理は省略しており，**4.2** の説明で使用したアルゴリズム中の関数 $f_0$，$f_1$，$t$，$g$，$h$ はそれぞれ $max$ で用いられる関数 $max$，$id$，$id$，$ge$，$le$ で置き換えられている．

次にトップダウン処理とボトムアップ処理の混在する関数の例として以下の関数 $sms$ を考える．

$$sms = \mathrm{HDF}\ (|)\ square\ sms\ sum\ ge\ le\ +\ +$$

$sms$ の定義中で呼び出されている $sum$ はパワーリ

```
1: global variable: n, r
2: local variable: i, x_i, f tmp
3: procedure max
4:     if (f = id) ∨ (r = n) then
5:         id (x_i)
6:         exit
7:     end if
8:
9:     send_recv(x_i, tmp, n − r − 1)
10:
11:    if b(i)[n − r − 1] = 0 then
12:        x_i ← ge (x_i, tmp)
13:        f ← max
14:    else
15:        x_i ← le (tmp, x_i)
16:        f ← id
17:    end if
18: end procedure
```

図 5　$max$ アルゴリズム
Fig. 5　$max$ algorithm.

スト要素の総和をパワーリストにする関数であり，$sum\ \langle 4\ 1\ 3\ 7\rangle = \langle 15\ 15\ 15\ 15\rangle$ である．

$$sum = \text{HDF}\ (|)\ id\ sum\ sum\ \pi_0\ \pi_1\ +\ +$$

$sum$ の前調節関数が $\pi_0$，$\pi_1$ であるので，$sum$ は HDF ボトムアップ関数である．$sms$ はパワーリスト要素の総和を求めるが，総和に含まれる最大要素のみがスカラ関数 $square\ \langle x\rangle = \langle x^2\rangle$ で 2 乗されている．例えばパワーリスト $\langle 4\ 1\ 3\ 7\rangle$ に対して $4+1+3+7^2 = 57$ なので，$sms\ \langle 4\ 1\ 3\ 7\rangle = \langle 57\ 57\ 57\ 57\rangle$ である．

$sms$ は再帰呼出し以外の関数呼出し（$sum$）を含んでいるために Achatz らの手法では記述できない．

HDF による $sms$ の記述について，定理 6 より $sms = sms^{\uparrow} \circ sms^{\downarrow}$ となる関数が存在して，次のように記述できる．

$$sms^{\downarrow} = \text{HDF}^{\downarrow}\ (|)\ square\ sms^{\downarrow}\ id\ ge\ le$$
$$sms^{\uparrow} = \text{HDF}^{\uparrow}\ (|)\ id\ sms^{\uparrow}\ sum\ +\ +$$

ここで，$sms^{\downarrow}$，$sms^{\uparrow}$ の左部分式への適用関数は $sms$ のそれが $sms$ 自身であることから，それぞれ $sms^{\downarrow}$，$sms^{\uparrow}$ 自身となっている．また右部分式への適用関数は $sms$ では $sum$ であるので，それぞれ $sum^{\downarrow} = id$，$sum^{\uparrow} = sum$ となっている．

関数 $sms^{\downarrow}$ と $sms^{\uparrow}$ は HDF トップダウン/ ボトムアップ関数なので，$sms = sms^{\uparrow} \circ sms^{\downarrow}$ に対応したハイパキューブアルゴリズムが生成できる．特に $sms^{\downarrow}$ に関して，自明な関数が $square$ であること以外は関数 $max$ と定義が一致しているので，図 5 の 5 行目にある $id\ (x_i)$ を $square\ (x_i)$ で置き換えたアルゴリズムが得られる．

### 5.2　CDF の記述例

スカラ関数を用いる不整合な関数の例として，実用上有用な高速フーリエ変換（FFT）を挙げる．関数 $FFT$ は離散化された信号を要素とするパワーリストを入力として，高速フーリエ変換された結果を返す関数である．関数 $FFT$ の定義は Misra [8]，Achatz ら [2] によって与えられている．その定義を簡単に説明する．ただし定義は本論文の記法で書き直してある．

$$FFT = \text{CDF}\ (\bowtie)\ id\ FFT\ FFT\ \pi_0\ \pi_1\ \oplus\ \ominus$$
$$FFT\ \langle x\rangle = \langle x\rangle$$
$$FFT\ (p \bowtie q) = (\oplus\ v\ w)\ |\ (\ominus\ v\ w)$$
$$\textbf{where}\ (v,\ w) = (FFT\ p,\ FFT\ q)$$

ここで $\oplus$，$\ominus$ は，それぞれ $\oplus\ v\ w = +\ v\ (*\ u\ w)$，$\ominus\ v\ w = -\ v\ (*\ u\ w)$ であるスカラ関数である．$u$ は 1 の原始根のべき乗を要素とするパワーリストであり，$u = \langle \omega^0\ \omega^1\ \ldots \omega^{\#p-1}\rangle$ $(\omega = \exp(-j\pi/\#p))$ である．各ノードは $u$ において自分に割り当てられた要素 $\omega^i$ を知っている必要があるが，これはノード番号と現在のラウンドから求められる．

定理 9 より，$FFT$ は，CDF トップダウン関数 $f^{\downarrow} = \text{CDF}^{\downarrow}\ (\bowtie)\ id\ f^{\downarrow}\ f^{\downarrow}\ \pi_0\ \pi_1 = inv$ と HDF ボトムアップ関数 $g^{\uparrow} = HDF^{\uparrow}\ (|)\ id\ g^{\uparrow}\ g^{\uparrow}\ \oplus\ \ominus$ に分解できる．また，定理 12 より，$f_H = inv \circ f^{\downarrow} = inv \circ inv = id$ として，$FFT = g^{\uparrow} \circ inv \circ f_H = g^{\uparrow} \circ inv \circ id$ が成り立つ．つまり $FFT$ のアルゴリズムは $id$，$inv$，$g^{\uparrow}$ のアルゴリズムの合成であり，それらは基本関数及び HDF ボトムアップ関数なのでアルゴリズムの生成が可能である．

$FFT$ は分割統治型のボトムアップ関数なので Achatz らの手法で記述可能である．しかし $g$，$h$ を $\pi_0$，$\pi_1$ ではなく任意のスカラ関数としたような定義 $FFT' = \text{CDF}\ (\bowtie)\ id\ FFT'\ FFT'\ g\ h\ \oplus\ \ominus$ では，分割統治型であるがトップダウン，ボトムアップ処理が混在しているため，そのままの形で Achatz らの手法を用いることはできない．我々の手法では上記の関数に加えて，非分割統治型の関数についても記述可能であり，ハイパキューブ上で実現できる．

## 6. むすび

本論文ではパワーリストによる並列アルゴリズムの仕様記述からハイパキューブアルゴリズムを生成するために，分解可能関数と呼ぶパワーリスト関数のクラ

スを提案した．分解可能関数は恒等関数と inversion 関数を基本操作とし，より多くのパワーリスト関数を記述できるように相互結合網をハイパキューブに制限することで高階関数 DC を拡張したものである．そして，分解可能関数のハイパキューブアルゴリズムを与え，その時間計算量は $O(n)$ であることを示した．

## 文　献


[1] K. Achatz and W. Schulte, “Architecture independent massive parallelization of divide-and-conquer algorithms,” Tech. Rep., Universiät Ulm, Fakulät für Infomatik, 1995.

[2] K. Achatz and W. Schulte, “Massive parallelization of divide-and-conquer algorithms over powerlists,” Science of Computer Programming, vol.26, no.1-3, pp.59–78, 1996.

[3] W. Adams, “Verifying adder circuits using powerlists,” Tech. Rep. CS-TR-94-02, University of Texas at Austin, Department of Computer Sciences, 1994.

[4] R. Gamboa, “The correctness of the fast fourier transform: A structured proof in acl2,” Formal Methods in System Design, vol.20, no.1, pp.91–106, 2002.

[5] D. Kapur and M. Subramaniam, “Automated reasoning about parallel algorithms using powerlists,” Lecture Notes in Computer Science, vol.936, pp.416–430, 1995.

[6] J. Kornerup, “Mapping a functional notation for parallel programs onto hypercubes,” Inf. Process. Lett., vol.53, no.3, pp.153–158, 1995.

[7] J. Kornerup, “Odd-even sort in powerlists,” Inf. Process. Lett., vol.61, no.1, pp.15–24, 1997.

[8] J. Misra, “Powerlist: A structure for parallel recursion,” ACM Trans. Programming Languages and Systems, vol.16, no.6, pp.1737–1767, 1994.

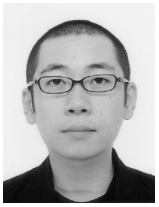
[9] J. Misra, “Generating-functions of interconnection networks,” Millennial Perspectives in Computer Science: Proc. 1999 Oxford-Microsoft Symposium in Honour of Sir Tony Hoare, ed. B.R.J. Davies and J. Woodcock, Prentice-Hall, Englewood Cliffs, 2000, http://www.cs.utexas.edu/users/psp/GeneratingFunctionsOfNetworks.pdf.

[10] J. Misra, “Derivation of a parallel string matching algorithm,” Inf. Process. Lett., vol.85, no.5, pp.255–260, 2003.

[11] D. Nassimi and S. Sahni, “Optimal BPC permutations on a cube connected SIMD computer,” IEEE Trans. Comput., vol.31, no.4, pp.338–341, 1982.

[12] V. Niculescu, “Parallel algorithms for fast fourier transformation using powerlist, parlist and plist theories,” Euro-Par 2002. Parallel Processing : 8th International Euro-Par Conference, ed. B. Monien and R. Feldmann, pp.400–404, Springer-Verlag, 2002. preprint version.


（平成 19 年 2 月 21 日受付，6 月 15 日再受付）




**東大野雅之**　（学生員）

平 15 岩手大大学院工学研究科博士前期課程了．現在，同大学院工学研究科電子情報工学専攻博士後期課程在学中．並列アルゴリズム，形式的仕様記述に関する研究に興味をもつ．



**西谷　泰昭**　（正員）

昭 50 東北大・工・電気卒．昭 56 同大大学院博士課程（情報）中退．同年日本電気（株）ソフトウェア生産技術研究所入社．昭 62 群馬大・工・助教授．平 12 岩手大・工・教授，現在に至る．分散アルゴリズム，ソフトウェア工学の研究に従事．工博．情報処理学会会員．