

# 修士論文

形状自在計算機システムにおける  
ネットワークインタフェースの研究

令和7年1月23日提出

指導教員  
入江 英嗣 教授

情報理工学系研究科

48-226422 後藤 俊樹

# 概要

新たな計算機の形として形状自在計算機システムが提案されている。形状自在計算機システムは多数のノードが無線通信により接続されたシステムである。このシステムでは、様々な形状を簡単かつ動的に作成することができる。

多数のチップの通信にはチップ間の通信ネットワークが不可欠である。形状自在計算機システムは物理層の特性やワークロードが従来の計算機システムと異なるため、従来のネットワークインターフェースをそのまま利用することは難しい。そこで形状自在計算機システムの物理層の特性やワークロードに合わせた、アドホックな無線ネットワーク構築手法やルーティングプロトコルが提案されてきた。

しかし、これらの提案ではソフトウェアシミュレーションによる評価が行われているに過ぎない。このためマイクロアーキテクチャレベルでの提案はされてこなかった。また、これらプロトコルは詳細な検証が行われておらず、問題が生じるシナリオが存在する。

本研究では、形状自在計算機システムのためのネットワークインターフェースのモデル作成、マイクロアーキテクチャの設計、RTL レベルの実装を初めて提案した。また作成したモデルに対する検証や、RTL レベルでの実装に対する消費電力や動作速度、実装面積等の評価を行った。これらの結果を考察し、形状自在計算機システムでのネットワークインターフェースの設計指針を提案する。

# 目次

第1章	はじめに	5
第2章	基礎知識	7
2.1	概要	7
2.2	Network On Chip の基礎知識	7
2.2.1	Network On Chip とは	7
2.2.2	Network On Chip のフローコントロール	7
2.2.3	Network On Chip のトポロジ	9
2.2.4	Network On Chip のルーティング	9
2.2.5	デッドロック	10
2.3	モデル検証	10
2.3.1	概要	10
2.3.2	有限状態機械と受理状態	10
2.3.3	線形時相論理	11
第3章	関連研究	14
3.1	形状自在計算機システムのアーキテクチャ	14
3.1.1	物理層	14
3.1.2	データリンク層	14
3.1.3	ネットワーク層	16
3.2	形状自在計算機システムにおけるネットワークプロトコル	17
3.2.1	概要	17
3.2.2	up*/down*ルーティング	18
3.2.3	ネットワーク参加プロセス	18
3.2.4	マルチツリーネットワーク参加プロセス	19
3.2.5	切断検知アルゴリズム	19
3.2.6	木構造を保持した動的再構成プロトコル	20
3.2.7	システム分離可能なネットワークプロトコル	21
3.3	座標推定アルゴリズム	21
3.3.1	概要	21
3.3.2	座標推定アルゴリズムの分類	21
3.3.3	グラフ理論における物理配置問題	23

<b>第 4 章</b>	<b>形状自在計算機ネットワークの問題点と形式化</b>	<b>25</b>
4.1	概要	25
4.2	各種用語の定義	25
4.3	デッドロックの発生	27
4.3.1	問題の概要	27
4.3.2	問題の定義	27
4.4	隣接テーブルの不整合	27
4.4.1	問題の概要	27
4.4.2	問題の定義	27
4.4.3	LTL による表現	28
4.5	ルーティングテーブルの不整合	28
4.5.1	問題の概要	28
4.5.2	問題の定義	28
4.5.3	LTL による表現	29
4.6	ノード id に関する問題	29
4.6.1	問題の概要	29
4.6.2	問題の定義	30
4.6.3	LTL による表現	30
4.7	同一の情報が複数回処理される問題	31
4.7.1	問題の概要	31
4.7.2	問題の定義	31
4.7.3	LTL による表現	32
<b>第 5 章</b>	<b>形状自在計算機のためのネットワークインターフェースの提案</b>	<b>33</b>
5.1	概要	33
5.2	フリット及びパケットの構成	33
5.3	リンク層	33
5.3.1	概要	33
5.3.2	構成図	34
5.3.3	再送処理	35
5.3.4	衝突回避	36
5.4	ネットワーク層	37
5.4.1	概要	37
5.4.2	構成図	38
5.4.3	ネットワーク参加処理	38
5.4.4	heartbeat 処理	40
5.4.5	再構成可能な動的再構成ネットワークプロトコル	41
5.4.6	木構造を保持した動的再構成ネットワークプロトコル	43
5.5	座標推定	48
5.5.1	概要	48
5.5.2	アルゴリズム	48
5.5.3	証明	49

<b>第 6 章</b>	<b>形状自在計算機のためのネットワークインタフェースの検証</b>	<b>50</b>
6.1	概要 . . . . .	50
6.2	検証 . . . . .	50
6.2.1	検証環境 . . . . .	50
6.2.2	検証モデル . . . . .	50
6.2.3	検証条件 . . . . .	51
6.2.4	検証結果 . . . . .	51
<b>第 7 章</b>	<b>形状自在計算機のためのネットワークインタフェースの評価</b>	<b>53</b>
7.1	概要 . . . . .	53
7.2	評価 . . . . .	53
7.2.1	評価項目 . . . . .	53
7.2.2	評価環境 . . . . .	54
7.2.3	ネットワーク初期化処理の定性的な評価 . . . . .	54
7.2.4	評価結果と考察 . . . . .	55
<b>第 8 章</b>	<b>本論文のまとめと今後の展望</b>	<b>58</b>
8.1	まとめ . . . . .	58
8.2	今後の展望 . . . . .	58

# 第1章 はじめに

半導体の微細化や高集積化により、計算機システムは小型化、高速化が進んだ。それと並行し、ソフトウェア開発における技術革新によって、より簡単に、より広く、より多様なアプリケーションを実行できるようになった。この結果、現代の社会では至る所に計算機が組み込まれ、人々の生活を支えている。

次世代の計算機の形として、形状変化が可能な計算機デバイスが注目されている [1]。現在実在する計算機デバイスは、形状が固定されており、計算機を利用する際には決まった形状に合わせて利用する必要がある。これに対し、形状変化可能な計算機デバイスは、人間や環境に合わせて形状を変化させることができる柔軟性をもつ。これにより、より普遍的なデバイスおよびアプリケーションが実現できることが期待される [2]。形状変化が可能な計算機システムのアプリケーションとして、マイクロロボットや形状変化するユーザーインターフェースが挙げられる。しかし、形状変更が可能な計算機の最適な実装方針は、未だに解決されていない課題である。

このような形状自在な計算機実現手法の一つの提案として、形状自在計算機システムが注目されている [3]。この計算機システムは複数の様々な用途に特化したチップが、隣接ノード間のワイヤレス通信によって協調し成り立つシステムである。無線通信により構成されているため、システム全体の形状を自在に変化させることができる。また、チップ同士で通信しあうことで、チップ同士が連携し、計算を行うことができる。

形状自在計算機システムのチップ間における無線通信にはオンチップコイルを使った誘導結合通信が採用されることが多い。誘導結合通信は、Ser/Des 回路を簡単に実装可能であり、かつ、高速な通信が可能であるが、隣接するチップに対してブロードキャスト通信が行われる。この制約により、既存の無線ネットワークプロトコルをそのまま適用すると非常に非効率な通信が行われることになる。このため、形状自在計算機システムに向けた新たなネットワークプロトコルが提案されてきた。[4,5]

しかし、提案されたネットワークプロトコルに関する評価は、ソフトウェア上のシミュレーションに留まっており、実際のハードウェア実装には至っていない。このため、実装はあくまで概案にとどまっており、マイクロアーキテクチャレベルの実装は行われておらず、詳細な実装方針や実装面積、消費電力、クロック周波数等のハードウェア特性に対する評価が不十分であった。また、ネットワークプロトコルの検証を行っておらず、ネットワークプロトコルの設計において問題点や曖昧な点があることが分かった。

このような背景から、本研究では、ネットワークインターフェースのモデル作成、マイクロアーキテクチャ設計、RTL レベルでの実装を行った。SPIN [6] による作成したモデルに対する部分的なモデル検証や、RTL レベルでの実装における消費電力や動作速度、面積といったハードウェアに関する評価を行った。以上の評価結果から、形状自在計算機システムにおけるネットワークインターフェース設計の指針と今後の展望について述べる。

本論文の構成は以下の通りである。第2章では、NoCの基礎知識や検証のための論理学等の背景知識について述べる。第3章では、形状自在計算機システムの関連研究や、座標推定に関する関連研究について述べる。第4章以降が本研究の内容になる。第4章では、形式検証のための問題の形式化について述べる。第5章では、ネットワークインターフェースのマイクロアーキテクチャ設計について述べる。第6章では、提案したネットワークインターフェースのモデルに対する検証結果について述べる。第7章では、提案したネットワークインターフェースのRTL実装の評価結果について述べる。第8章では、評価結果に基づいて、今後の展望と本論文のまとめについて述べる。

## 第2章 基礎知識

### 2.1 概要

形状自在計算機システムは、多数の計算機が通信し協調して計算を行う分散システムである。このようなシステムを設計するためには、ネットワークプロトコルの設計が重要である。よいネットワークプロトコルの設計には、効率と正しさの二つの要素が必要である。

多数の計算機が協調するシステムにおける効率的なネットワーク技術として、ネットワークオンチップ (NoC) がある。NoC は、複数のプロセッサコアやメモリモジュールなどのコンポーネントを、チップ上で相互接続するための通信インフラストラクチャである。NoC の技術を活用することで、多数の計算機が協調する形状自在計算機システムのネットワークプロトコルを設計することができる。

設計したネットワークプロトコルが正しいことを保証するためには、モデル検証が有効である。モデル検証は、システムのモデルを作成し、そのモデルが仕様を満たしているかどうかを検証する技術である。本論文では、SPIN を用いたモデル検証を行う。

本章では、本論文で用いるネットワークオンチップ (NoC) の基礎知識と、モデル検証の基礎知識について説明する。

### 2.2 Network On Chip の基礎知識

#### 2.2.1 Network On Chip とは

Network On Chip (NoC) とは、複数のプロセッサコアやメモリモジュールなどのコンポーネントを、チップ上で相互接続するための通信インフラストラクチャである。従来のバス型アーキテクチャでは、スケーラビリティに大きな問題点があった。プロセッサコアの数が増加すると、配置配線の問題が大きくなり、性能が低下する。NoC はルータとリンクを用いて、複数のコアを接続することで、スケーラビリティを向上させることができる。このため、ムーアの法則による性能向上が難しくなるにつれて、NoC は、多コアプロセッサや多コア SoC の通信インフラストラクチャとして注目されている [7]。

このセクションでは、NoC の基本的な構成要素と分類について説明する。分類には、トポロジ、ルーティング、フローコントロールが含まれる。

#### 2.2.2 Network On Chip のフローコントロール

NoC のフローコントロールは、データの転送を効率的に行うための技術である。ネットワークの論理的な単位であるパケットは一般に可変長であり、また転送の単位としては大



きく扱いづらい。このため、パケットをフリットと呼ばれる固定長のデータに分割し、転送することが一般的である [8]。

分割されたフリットの転送は大きくバッファを持つ方式と持たない方式で2分される。まずバッファを持たない方式では以下のような方式がある。

### 1. 単純方式

受け取ったフリットをそのまま次のノードに送信する方式。

### 2. サーキットスイッチング方式

最終ノードまでセットアップリクエスト、及びそのレスポンスを送受信することで経路を確保し、その後フリットを送信する方式。

単純方式では受信側での準備が整っていない場合、フリットを破棄することになる。このためパケットロスにより非常に効率が悪い場合がありえる。

サーキットスイッチング方式では、経路をあらかじめ確保することで、フリットの破棄を防ぐことができる。しかし、この方式も経路確立までのオーバーヘッドが大きく、一般的にはあまり使われることはない。また、動的な接続が頻繁に起こるような経路の信頼性が低い場合では、確立した経路を維持することが難しく、不向きであると考えられる。

次にバッファや配置に関するフリット転送方式を3種類紹介する。これらの転送方式は、転送において、どのようなバッファを持つか、どのようなフリットを保持するかの違いがある。

#### 1. ストアアンドフォワード方式

パケット全体を格納できるバッファを持ち、パケット全体を受信した後に送信する方式。

#### 2. ワームホール方式

フリットを保持できるバッファをもち、フリットを受信した後に送信する方式。

#### 3. ヴァーチャルカットスルー方式

パケット全体を格納できるバッファを持ち、パケットの一部を受信した後に送信する方式。

一般にパケット全体を保持する方式では全体の情報が使えるため処理は簡単になるが、回路面積が大きくなる。また、パケット長の最大値がバッファのエントリによって制約される。全体を持たず、フリットが様々なノードに分散している場合、処理をパイプライン化できるため、スループットが向上する。しかし、動的な経路変更の際には、分散したフリットを集める必要があるため、処理が複雑化することやエントリ数増大の問題がある。

その他 VC(Virtual Channel) を用いた転送方式もある。VC を用いることで、フリットの転送を複数のチャンネルで行うことができ、経路の多重化等、様々な効率化や信頼性向上が期待される。しかし VC を用いた方法は、回路面積が大きくなることや、複雑な制御が必要になる問題点がある。

### 2.2.3 Network On Chip のトポロジ

NoC のトポロジは、ネットワーク内のルータとリンクの配置を表し、ネットワークの性能や信頼性に大きな影響を与える。トポロジは、大きくレギュラーネットワーク、イレギュラーネットワークに分類される。レギュラーネットワークは、ルータとリンクが規則的に配置されているネットワークであり、長方形などがある。規則的に配置されているため、形状に即した最適化や性能予測が可能である。対して、イレギュラーなネットワークは、ルータとリンクが規則的でないネットワークであり、複雑な形状を持つ。不規則なネットワークでは事前に形状を利用した計算ができず、かつどのような形状でも可能なアルゴリズムが必要である。

### 2.2.4 Network On Chip のルーティング

ルーティングには、以下の大きく 3 種類ある。

#### 1. 決定的なルーティング

ルーティングアルゴリズムが出力する候補の経路の個数が一つであるルーティングのことで、ソースから目的地への経路が一意に定まるという特性がある。

#### 2. オブリーブiasルーティング

候補の経路の個数が複数のルーティングで、経路の選択アルゴリズムには、現在の混雑度など、ネットワークステータスを考慮しない。

#### 3. アダプティブルーティング

現在の混雑度など、ネットワークステータスを考慮した経路選択をすることができる。事前の解析だけではなく、現在の情報を用いることが可能であるため、効率上は有利であるが、ハードウェアやソフトウェアが複雑になるというトレードオフがある。

決定的なルーティングは一意に定まるため、デットロックへの対策が容易である。しかし、特定の経路が混雑してしまうと、その経路がボトルネックとなり、ネットワーク全体の性能が低下する可能性がある。オブリーブiasルーティングは、経路を複数もつため、負荷の分散が可能である。しかし、決定的なルーティング同様にネットワーク全体の性能を考慮しないため、ネットワーク全体の性能が低下する可能性がある。アダプティブルーティングは、ネットワーク全体の性能を考慮した経路選択が可能であるため、ネットワーク全体の性能を最適化することができる。しかし、ネットワーク全体の性能を考慮するため、ハードウェアやソフトウェアが複雑になるというトレードオフがある。

障害が発生した時の回復方法には以下の 2 つがある。

#### 1. 静的再構成

ネットワーク再構成時に古いルーティング情報のパケットを完全に破棄する方式

#### 2. 動的再構成

ネットワーク再構成時に古いルーティング情報を新しいルーティング情報に変換し、古いルーティング情報をもったパケットであっても破棄せずに転送する方式

動的な方法では、古いルーティング情報をもったパケットを新しいルーティング情報に変換する必要があるため、ハードウェアやソフトウェアが複雑になるというトレードオフがある。しかし、パケットロスを防ぐことができるメリットがある。

### 2.2.5 デッドロック

デッドロックとは、複数のプロセスが相互にリソースを待ち合わせることで、処理を進めることができなくなる状態のことである。Dally の理論により、デッドロックが発生しない必要十分条件が示されている [9]。

**Theorem 2.1** (デッドロックの必要十分条件). ルーティング関数がデッドロックが発生しないための必要十分条件は、チャンネル依存性グラフ (*Channel Dependency Graph, CDG*) がサイクルを持たないことである。

## 2.3 モデル検証

### 2.3.1 概要

システムが正しいことを保証するためには、形式検証が有効である。形式検証は、システムの仕様を数学的に記述し、その仕様が満たされているかどうかを検証する技術である [10]。形式検証は、システムの信頼性を向上させるために産業界を初め、広く利用されている。

形式検証の一つとしてモデル検証がある。モデル検証は、システムのモデルを作成し、そのモデルが仕様を満たしているかどうかを検証する技術である。

本セクションでは、モデル検証の基本的な概念と、モデル検証で用いられる線形時相論理について説明する。

### 2.3.2 有限状態機械と受理状態

#### 概要

検証対象のシステムは、入力により刻々と変化する。この変化を数学的に形式化するため、状態機械を用いる。状態機械は、状態と遷移の集合で構成され、初期状態から入力を受け取ることで順に遷移していく。

システムが正しいとは、最終的な遷移が受理条件を満たすことである。このため、いかなる遷移であっても受理条件をみたすことが証明できれば、システムにバグがないことが証明できる。

まずは、有限の状態をもつ機械について説明する。とりうる状態が有限であっても、永続的な状態列をもつ場合がある。無限の状態列にたいして終了状態を明確に定義することが難しい。この様な場合に、受理状態がどのように定義されるかについて説明する。

## 有限状態機械

SPIN では有限状態機械 (Finite State Machine, FSM) を用いてシステムのモデルを記述する。有限状態機械は、以下の 5 つの要素で構成される。

**Definition 2.1** (FSM). 有限状態機械 (Finite State Machine, FSM) は、5 つの要素  $M = (S, s_0, \Sigma, T, F)$  で構成される。

1.  $S$  は有限状態集合
2.  $s_0 \in S$  は初期状態
3.  $\Sigma$  は入力アルファベット
4.  $T : S \times \Sigma \rightarrow S$  は遷移関数
5.  $F$  は終了状態集合

有限状態機械は、初期状態から入力アルファベットで定義される入力を受け取り、遷移関数によって次の状態に遷移する。遷移ができない場合、最終的な状態が終了状態に含まれる時、その遷移は受理される。このような有限状態機械の実行は状態と入力の列として表現される。

**Definition 2.2** (実行). FSM  $M$  の実行は以下を満たす列である。状態列は無限列である場合もあり、その場合は  $\omega$ -run と呼ばれる。

$$((s_0, i_0), (s_1, i_1), (s_2, i_2), \dots), s_i \in S, i_i \in \Sigma \quad (2.1)$$

ただし、

1.  $s_0$  は初期状態
2.  $s_{i+1} = T(s_i, i_i)$

## 受理状態

実行が有限列である場合の受理状態の定義は簡単であるが、無限列の場合は難しい。無限列の場合の受理条件を、ここでは、Buchi の受理条件を用いて定義する [11]。

**Definition 2.3** (Buchi の受理条件).  $\omega$ -run が Buchi の受理条件を満たすとは、無限回繰り返される状態が受理状態に含まれることである。すなわち、 $\sigma$  を  $\omega$ -run とし、 $\sigma$  のなかで無限回登場する状態の集合を  $\sigma'$  とすると、 $\sigma' \cap F \neq \emptyset$  である。

すなわち、無限列においては受理状態が無限回登場する場合、その列は受理される。

### 2.3.3 線形時相論理

#### 概要

線形時相論理 (Liner Temporal Logic, LTL) は、数理論理学の一分野である。LTL は、命題論理と様相論理を組み合わせた論理体系であり、時間の遷移を数学的に記述することができる。このため、モデル検証でよく用いられる論理体系である。ここでは LTL の基本的な構成要素と、LTL の意味論について説明する。

## 線形時相論理 (LTL)

LTL は、時間的な制約を表現するための論理体系である。LTL では命題論理で用いる  $\neg, \wedge, \vee, \rightarrow$  等の記号に加えて、様相演算子  $\Box$ (常に)、 $\Diamond$ (ある時点で)、時間演算子  $X$ (次の時点で)、 $U$ (まで)、 $R$ (解放) を用いる。まず、LTL の基本的な構成要素である論理式について帰納的に定義する。

**Definition 2.4** (論理式). 命題変数の集合を  $Prop$  とし、命題結合子  $\neg, \wedge, \vee, \rightarrow$ 、様相演算子  $\Box, \Diamond$ 、時間演算子  $X, U, R$  を用いて、LTL の論理式を次のように定義する。ここで  $p \in Prop$  は命題変数を表し、 $\varphi, \psi$  は LTL の論理式を表す。

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \rightarrow \psi \mid \Box\varphi \mid \Diamond\varphi \mid X\varphi \mid \varphi U\psi \mid \varphi R\psi \quad (2.2)$$

続いて、論理式が成り立つということを定義するため、論理式の最小単位である命題変数の意味を定義する。命題変数の成立を定義するため、付値関数を導入する。付値関数は、命題変数に対して、その命題変数が成立する状態の集合を返す関数である。

**Definition 2.5** (付値関数). 付値関数  $V$  は、命題変数に対して、状態集合の部分集合を返す関数である。

$$V : Prop \rightarrow 2^S \quad (2.3)$$

$s \in V(p)$  のとき、 $s$  で  $p$  が成立するという。

命題変数の成立が定義できたため、論理式の意味論を定義する。

**Definition 2.6** (論理式の意味論). 線形時相論理 (LTL) の意味論は、無限の状態列  $\sigma = s_0, s_1, s_2, \dots, s_k, \dots$  と、その中のある時点  $k \geq 0$  における論理式  $\varphi$  の満たされる条件を定義する。 $\sigma, k \models \varphi$  で、状態列  $\sigma$  の時点  $k$  において  $\varphi$  が成立することを表す。

$$\sigma, k \models p \in Props \iff s_k \in V(p) \quad (2.4)$$

$$\sigma, k \models \neg\varphi \iff \sigma, k \not\models \varphi \quad (2.5)$$

$$\sigma, k \models \varphi \wedge \psi \iff \sigma, k \models \varphi \text{ かつ } \sigma, k \models \psi \quad (2.6)$$

$$\sigma, k \models \varphi \vee \psi \iff \sigma, k \models \varphi \text{ または } \sigma, k \models \psi \quad (2.7)$$

$$\sigma, k \models \varphi \rightarrow \psi \iff \sigma, k \not\models \varphi \text{ または } \sigma, k \models \psi \quad (2.8)$$

$$\sigma, k \models \Box\varphi \iff \forall j \geq k, \sigma, j \models \varphi \quad (2.9)$$

$$\sigma, k \models \Diamond\varphi \iff \exists j \geq k, \sigma, j \models \varphi \quad (2.10)$$

$$\sigma, k \models X\varphi \iff \sigma, k+1 \models \varphi \quad (2.11)$$

$$\sigma, k \models \varphi U\psi \iff \exists j \geq k, \sigma, j \models \psi \text{ かつ } \forall i (k \leq i < j), \sigma, i \models \varphi \quad (2.12)$$

$$\sigma, k \models \varphi R\psi \iff \forall j \geq k, \sigma, j \models \psi \text{ または } \exists i (k \leq i < j), \sigma, i \models \varphi \quad (2.13)$$

ただし、 $k$  を省略した  $\sigma \models \varphi$  は  $\sigma, 0 \models \varphi$  と同義である。

2.4 から 2.8 は命題論理の意味を定義している。2.4 は命題変数の意味を定義している。2.5 から 2.8 は、否定、論理積、論理和、含意の意味を定義している。

2.9 から 2.13 は、時相論理で出てくる記号の意味を定義している。2.9 は、 $\varphi$  が時刻  $k$  より後では常に成り立つことを表す。2.10 は、 $\varphi$  が時刻  $k$  より後で一度でも成り立つことを表す。2.11 は、 $\varphi$  が次の時刻で成り立つことを表す。2.12 は、時刻  $k$  より後で  $\psi$  が成り立つまで  $\varphi$  が常に成り立つことを表す。2.13 は、時刻  $k$  より後のある時刻で  $\psi$  が成り立たないならば、その時刻以前で  $\varphi$  が成り立つことを表す。

LTL の論理式が等しいことを定義する。

**Definition 2.7** (論理式の等価性). LTL の論理式  $\varphi, \psi$  が同値であるまたは意味的に同値であるとは、任意の状態列  $\sigma$  に対して  $\sigma \models \varphi$  ならば  $\sigma \models \psi$  であり、逆も成り立つことを表す。

同値な LTL の例として以下のようなものがある。

$$\Box\varphi \equiv \neg\Diamond\neg\varphi \quad (2.14)$$

$$\Box\varphi \equiv \mathbf{True}U\varphi \quad (2.15)$$

$$\Diamond\varphi \equiv \neg\Box\neg\varphi \quad (2.16)$$

$$\Diamond\varphi \equiv \mathbf{False}R\varphi \quad (2.17)$$

$$\varphi R\psi \equiv \neg(\neg\varphi U \neg\psi) \quad (2.18)$$

$$\varphi U\psi \equiv \neg(\neg\varphi R \neg\psi) \quad (2.19)$$

$$\neg X\varphi \equiv X\neg\varphi \quad (2.20)$$

## 第3章 関連研究

### 3.1 形状自在計算機システムのアーキテクチャ

形状自在計算機システムは、既存のコンピュータにはない特徴を持つ。このため、このシステムに合ったネットワークアーキテクチャが提案されてきた。本セクションでは、これまでに提案されてきた形状自在計算機システムのアーキテクチャやネットワークプロトコルについて説明する。

#### 3.1.1 物理層

形状自在計算機システムは、隣接ノード間でワイヤレス通信を行うことで協調するシステムである。チップ間の無線通信には、オンチップコイル間の近接場誘導結合通信が用いられることが多い。このため、ここでは近接場誘導結合通信について説明する。

近接場誘導結合通信は、チップ間の無線通信を行うための技術である。標準 CMOS プロセスでの製造時に、内部配線としてチップの外周にそったコイルを配置することができる。このコイルによって、正方形のチップの場合、辺が隣接したコイル同士でワイヤレスバスを構築することができる [12]。

一般的な無線通信では、様々な周波数帯域を利用し周波数の多重化を行うことで、高速な通信を実現している。しかし、周波数帯域を生かした通信では、周波数の変調回路が必要となるためチップの面積が大きくなる。これに対して、近接場誘導結合通信は周波数変調を行わず Non-return-to-zero (NRZ) のベースバンド信号を直接伝送する。このため、回路は 3.1 に示すような簡単な構成で済み、チップの面積を小さくすることができる。

送信回路には H-Bridge 送信機、受信回路にはヒステリシスコンパレータが主に用いられる。H-Bridge 送信機は、コイルに流れる電流を制御することで、コイルに磁界を発生させる。発生した磁界は、隣接するコイルに誘導され、パルス状の電圧が発生する。受信回路では、発生したパルス状の電圧を検出し、ヒステリシスコンパレータで元々の NRZ 信号に復調する。45nm CMOS プロセスでの実装では、コイル径  $300\mu\text{m}$  である時、最大転送速度は 14.3Gb/s で、電力効率は 0.55pJ/b である [3]。

#### 3.1.2 データリンク層

形状自在計算機システムの受信回路は隣接するチップそれぞれに対してチャンネルが独立しておらず、すべてを同一のチャンネルとして扱う。このため隣接する異なるチップから送信された信号が衝突する可能性があり、データリンク層では衝突処理に対応するプロトコルが必要となる。

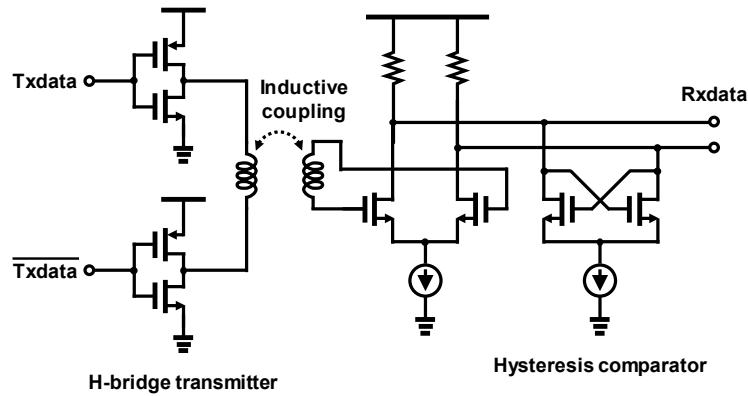


図 3.1: 近接場誘導結合通信の概念図 ([4] より引用)

従来の形状自在計算機システムでは衝突の検出、及び再送処理を行うプロトコルが提案されている。送信側のチップで衝突を検知した際と、Ack 信号を受信しなかった際に再送信を行う。

データリンクの全体図を 3.2 に示す。この回路には Ser/Des 回路、衝突検知回路、再送回路、検査回路がある。検知回路では、形状自在計算機では送信データが受信データにも入ることを利用して、衝突を検知する。伝送間隔の調整には CSMA/CD (Carrier Sense Multiple Access with Collision Detection) [13] 等のアルゴリズム同様に、べき乗バックオフを用いる。送信にはアドレスを指定したユニキャスト通信と、アドレスを指定しないブロードキャスト通信がある。ブロードキャスト通信は送信側が ACK 信号を処理することができないため、受信側からの ACK 信号を行わない。

以下がデータリンク層におけるデータ送信アルゴリズムの概略である。後に実は衝突検知回路が意味をなさないことを示す。



---

**Algorithm 1** データリンク層のデータ送信
 

---

```

1: txFlit  $\leftarrow$  sendBuffer
2: failedCount  $\leftarrow$  0
3: while failedCount < maxRetry do
4:   send(txFlit)
5:   if collision detected then failedCount  $\leftarrow$  failedCount + 1
6:     // Exponential backoff
7:     waitTime  $\leftarrow$  baseTime * (1 << failedCount) + random()
8:     sleep(waitTime)
9:   continue
10: end if
11: wait until received or timeout
12: rxFlit  $\leftarrow$  receive()
13: if rxFlit is ACK then
14:   break
15: end if
16: failedCount  $\leftarrow$  failedCount + 1
17: end while
  
```

---

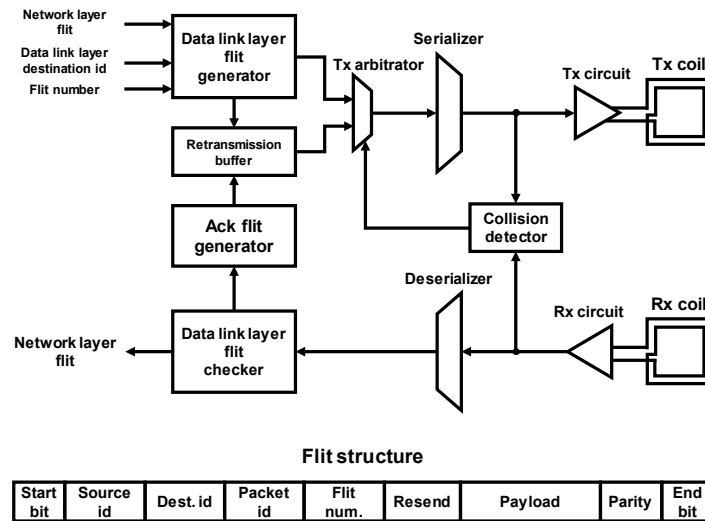


図 3.2: 従来の形状自在計算機システムのデータリンク層 ([4] より引用)

### 3.1.3 ネットワーク層

形状自在計算機システムでは、ノードの動的な形状変更があり形状が決まっていないため、イレギュラーなネットワークに対してネットワークグラフを動的に構成する必要がある。

ある。

ネットワーク層の情報の単位であるパケットは、固定長のデータであるフリットに分割される。フリットは Head, Data, Tail の 3 つの種類から構成される。

ネットワーク層全体の構成図を 3.3 に示す。ネットワーク層はルーティングテーブル、アドレス計算回路、ルーティング回路から構成される。ルーティングテーブルは、各ノードに対して次のノードのアドレスを保持する。アドレス計算回路では、アドレスが自身のものか、他のノードのものかを判定する。

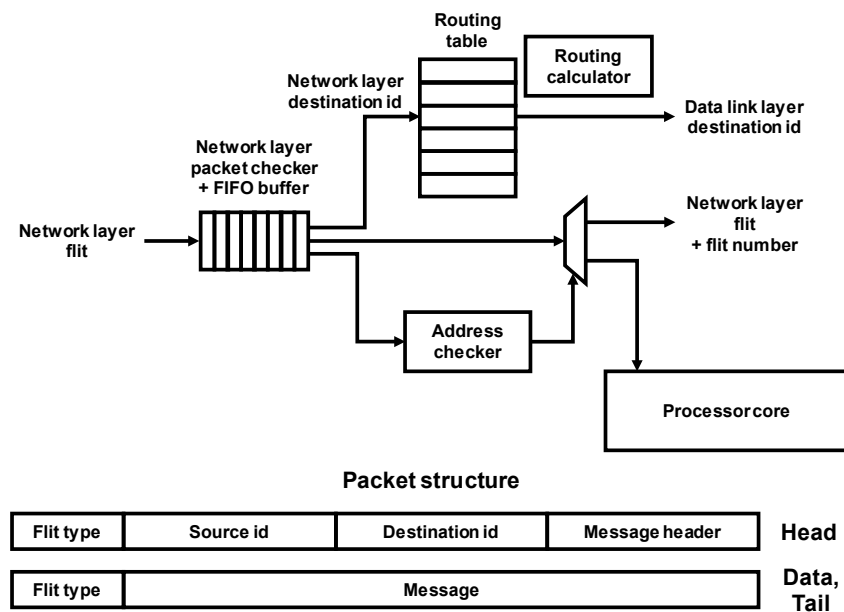


図 3.3: 従来の形状自在計算機システムのネットワーク層 ([4] より引用)

## 3.2 形状自在計算機システムにおけるネットワークプロトコル

### 3.2.1 概要

形状自在計算機システムでは形状が動的に変化する。このため、通信経路が静的な形で定まっている NoC のプロトコルは適用できない。そのため、形状自在計算機システムに適したネットワークプロトコルが提案されている。

提案されているネットワークプロトコルはいずれも、ルーティング経路として  $up^*/down^*$  ルーティング [14] を採用している。このため、 $up^*/down^*$  ルーティングについて説明した後、形状自在計算機システムのネットワークプロトコルで共通の処理である参加処理、切断検知処理を説明する。その後、切断後の再構成処理について説明する。

### 3.2.2 up\*/down\*ルーティング

up\*/down\*ルーティングはスパニングツリーを用いてルーティングを行う決定的なルーティングである。up, down 方向は以下のように定義される。ルートへ向かう方向が up 方向、ルートから離れる方向が down 方向である。

**Definition 3.1** (up, down 方向). あるノードの up 方向とは、ルートからの距離が減少する方向であり、down 方向とは、ルートからの距離が増加する方向である。

up down を用いたルーティングアルゴリズムを以下に示す。up\*/down\*ルーティングは、目的地を最初に子ノードに持つノードまで up 方向に移動したあと、目的地を持つ子ノードまで down 方向に移動するルーティングである。

---

**Algorithm 2** up\*/down\*ルーティング

---

```
1: currentNode ← self
2: while currentNode ≠ destinationNode do
3:   if destinationNode in currentNode.children then
4:     // down, find next node by routingTable
5:     currentNode ← routingTable(current.children, destinationNode)
6:   else
7:     // up, just use parent node
8:     currentNode ← current.parent
9:   end if
10: end while
```

---

上記のアルゴリズムより、同一のツリー内部であれば任意のノード間で通信が可能である。またルーティング経路に循環が存在しないため、どのような形状であってもデッドロックが発生しない。

up\*/down\*ルーティングの問題点としては、スパニングツリーによってネットワーク性能が左右されることが挙げられる。up\*/down\*ルーティングは、あるスパニングツリー上における最短の経路であるが、物理形状における最短経路とは異なる場合がある。このため、スパニングツリーによってネットワーク性能が左右される。また、ルートに近いノードに通信が集中するため、ホットスポットが発生しやすい。

### 3.2.3 ネットワーク参加プロセス

ネットワークの構成員はコーディネータとルータに分かれる。コーディネータはシステムに一つであり、形成されるスパニングツリーのルートノードになる。現在はコーディネータは静的に決定されているが、分散システムにおけるリーダー選出アルゴリズムを用いて動的に決定することも考えられる。

以下、提案されているネットワーク参加プロセスを示す [4]。参加しようとしているルータを  $r$ 、コーディネータを  $c$  とする。

1. ルータ  $r$  が一時的な id をランダムに生成する。

2. ルータ  $r$  が隣接ノードにたいして、親ノード要求 (Parent Request, preq) をブロードキャストする。
3. preq を受信した隣接ノードの中で既にネットワークに参加しているノードがあれば、親ノード承認 (Parent Ack, pack) を返す。
4. ルータ  $r$  は最初に受信した pack の送信元を親ノードとして選択する。
5. ルータ  $r$  は親ノードに対して、ネットワーク参加要求 (Join Request, jreq) を送信する。
6. ネットワーク参加済みノードは jreq を受信後、コーディネータまで up 方向に送信する。送信時にルーティングテーブルにエントリを追加する。
7. コーディネータは jreq を受信後、jreq を受け取った順番で真の id を割り当てる。
8. コーディネータはネットワーク参加承認 (Join Ack, jack) を送信する。
9. 経路上のルータは jreq 転送時のエントリを用いて jack を送信する。
10. ルータ  $r$  は jack を受信後、ネットワークに参加する。

### 3.2.4 マルチツリーネットワーク参加プロセス

マルチツリーは複数の仮想チャネルを用いて複数のスパニングツリーを形成するネットワークプロトコルである [4]。前述したとおり、up\*/down\* ルーティングでは、ホットスポットが発生しやすい問題があった。しかしマルチツリーでは複数のスパニングツリーを形成することで、ホットスポットを分散させることができる。マルチツリーではコーディネータが仮想チャネルの数だけ存在する。

マルチツリー構築はまだ初期化が済んでいないチャネルに対して、単一ツリーの参加プロセスをラウンドロビン形式で行うことで実現できる。すべてのツリー参加をもってネットワークの初期化完了とする。

マルチツリーは仮想チャネル分のバッファ、及びルーティングテーブルを持つ必要がある。このため、ハードウェアが複雑化する。また形状の変更による再参加時には、全ての仮想チャネルに対して再参加プロセスを行う必要があり、ネットワーク再構築に時間がかかる。

### 3.2.5 切断検知アルゴリズム

切断検知では Heartbeat Monitoring を用いて生存確認をおこなう。Heartbeat Monitoring は、定期的にノード間で信号を送信することで、一定期間信号を受信しなかった場合に切断と判断する手法である。親ノードが子ノードの切断を検知した場合は、切断されたことをブロードキャストの形でネットワーク全体に通知する。

切断されたノードの子ノードはネットワーク接続を失うため、再度ネットワークに参加する必要がある。次節で説明する動的再構成プロトコルを用いて再参加を行う。

### 3.2.6 木構造を保持した動的再構成プロトコル

up\*/down\*ルーティングはスパニングツリーを用いてルーティングを行うルーティングであるため、あるノード間で切断が発生した場合、部分木が生成される。木構造を保持した動的再構成プロトコルは、部分木を保持したままネットワーク全体の再構成を行うプロトコルである。再構成プロセスは探索と更新の2つのフェーズに分かれる。

切断されたスパニングツリーを再構成サブツリーと呼ぶ。以下、対象としている再構成サブツリーのルートノードを  $N_{rc}$  とし、 $N_{rc}$  をルートノードとするスパニングツリーに属するノードを  $SubTree(N_{rc})$  とする。

#### 探索フェーズ

$N_{rc}$  は、 $SubTree(N_{rc})$  に対して、再構成フラグをブロードキャストの形で送信する。その後、 $N_{rc}$  から行きがけ順に以下の復旧経路の探索を行う。

---

**Algorithm 3** 探索フェーズ

---

```
1: currentNode  $\leftarrow N_{rc}$ 
2: while True do
3:   retryCount  $\leftarrow 0$ 
4:   while retryCount < maxRetry do
5:     currentNode.broadcast(preq)
6:     if currentNode gets pack from not  $SubTree(N_{rc})$  then
7:        $N_{nrc} \leftarrow$  currentNode
8:       goto Update Phase
9:     end if
10:    retryCount  $\leftarrow$  retryCount + 1
11:   end while
12:   while True do
13:     nextNode  $\leftarrow$  currentNode.nextPreorderTraversal()
14:     if nextNode is null then
15:       if currentNode is  $N_{rc}$  then
16:         currentNode.resetTraversalState()
17:         break
18:       end if
19:       currentNode  $\leftarrow$  currentNode.parent
20:     else
21:       currentNode  $\leftarrow$  nextNode
22:       break
23:     end if
24:   end while
25: end while
```

---

## 更新フェーズ

$SubTree(N_{rc})$  のなかで pack を受信したノードを  $N_{nrc}$  とする。このとき経路の更新は以下のように行う。

---

**Algorithm 4** 更新フェーズ

---

```
1: currentNode  $\leftarrow N_{nrc}$ 
2: while currentNode  $\neq N_{rc}$  do
3:   oldParent.parent  $\leftarrow$  currentNode
4:   currentNode  $\leftarrow$  oldParent
5:   oldParent  $\leftarrow$  currentNode.parent
6: end while
```

---

### 3.2.7 システム分離可能なネットワークプロトコル

これまでのネットワークは分離されたネットワークは、コーディネータが存在するネットワークに再度参加するまで通信を行うことができなかった。システム分離可能とは分離されたネットワークがそれぞれ独立して動作できることを指す [5]。動的な再構成ではネットワークに復帰することが目的であったのに対して、システム分離可能なネットワークプロトコルでは、ネットワークから離脱してもシステムが維持されることが目的である。

## 3.3 座標推定アルゴリズム

### 3.3.1 概要

ディスプレイなど、分散した計算機がどの位置にあるのかを推定することが必要な場面がある。また、後に明らかにするが、形状自在計算機システムでは位置関係がわかっていることで衝突を回避することができる。座標推定アルゴリズムは、分散した計算機の相対的な位置関係を推定するアルゴリズムである。

本セクションでは、座標推定アルゴリズムの分類と、グラフを用いた座標推定アルゴリズムの関連研究について説明する。

### 3.3.2 座標推定アルゴリズムの分類

形状推定を行うモデルとして図 3.4 のような、システム外部の装置を用いるモデル (図 3.4a)、システム内部に小型デバイスの集合体の他に監視者を用いるモデル (図 3.4b、小型デバイスの集合体のみで構成されるモデル (図 3.4c の 3 通りが考えられる。

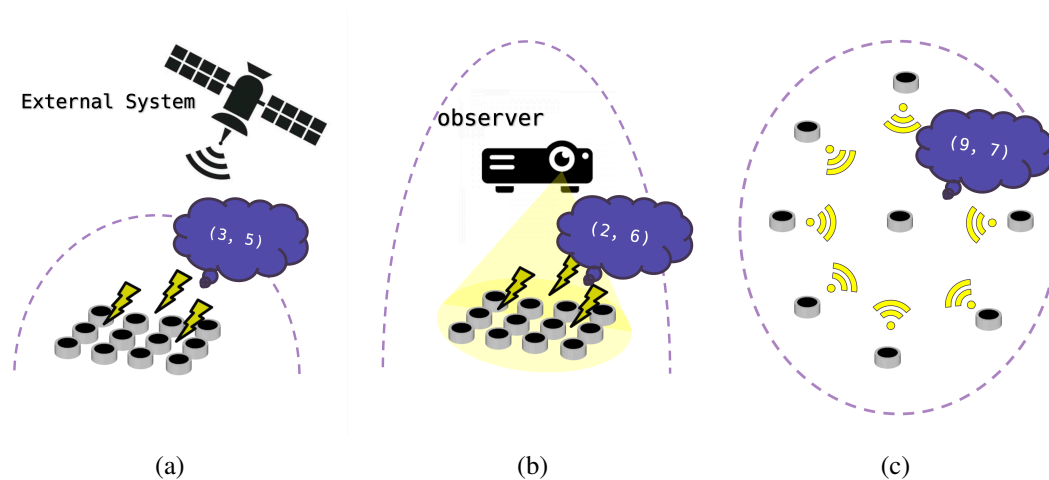


図 3.4: コレクティブユーザインタフェースにおける形状推定。(a) は GPS などの外部システムを利用して座標推定を行うモデル、(b) はシステム内部にカメラなどの監視者を取り入れ座標推定を行うモデル、(c) は素子間通信のみで座標推定を行うモデルを表す

### 外部システムを用いた位置推定

まず図 3.4a のようなシステム外部の装置を用いるモデルを考える。位置推定に用いることができる外部装置として、まず GPS(Global Positioning System) といったシステムが挙げられる。このような外部デバイスを用いる利点は特別なアルゴリズムを必要とせず非常に簡単に推定することができる。しかし GPS では最高でも数センチオーダの誤差が生まれる [15] のに加えて、通信間の障害物や、太陽フレアといった気象条件に対して脆弱であり、かつ受信回路のコストや回路面積を取るなど、小型計算機の集合体に対する位置推定として用いるのに相応しくない。

### 監視者を用いた位置推定

そこで環境の影響を防ぐため、かつ、より精度よく推定するために図 3.4b のようなシステム内部に監視者のようなデバイスを用いる手法が提案されている。このようなシステムでは、小型計算機の中での機能は最小限にし、集合体の外側にあるカメラやプロジェクタなどの機器を用いて位置推定を行う [16]。しかしこの手法では集合体の外側に機器を設置するコストや、監視者の観測できる範囲に限定されるという制約が存在する。このため、例えば手や足で物体を操作した際、それらが監視者の障害物となり、位置推定できず、結果としてインタラクションが阻害されること等も考えられる。

### 素子間通信のみでの位置推定

最後に考えられるモデルは図 3.4c 小型デバイスのみで構成されるモデルである。このモデルでは小型デバイス同士が協調し合うことでシステムのトポロジーを推定する。このような推定方法では大きく指向性のある通信とない通信で分けることができる。

まず指向性のある通信や他のノードの方角の取得が可能な小型デバイスを考える。このようなデバイスでは情報が来た方角を得られるため向きがわかる。加えて通信強度から距離の絶対値を計算できる。このため情報元のデバイスからのベクトルを求めることが可能になり、これを様々な素子同士で行うことで全体のトポロジを求めることができる。このような通信を用いている例として [17, 18] がある。しかし、指向性のある通信を実現するには高価なセンサが必要になり、かつそれを配置する面積も必要になる。

次に指向性のない通信を用いて位置推定を行う例を考える。指向性のない通信ではすでに座標が確定している複数点からの距離を強度情報から計算することで推定することができる。例えば2次元平面上の物体であれば3点からの距離をそれぞれ求めることで計算可能である。これを用いた例として [19] がある。しかしこの推定では3点からの情報を精度よく取得する必要がある、距離情報の誤差が問題になる。これを解決するための取り組みとして [20, 21] などが挙げられる。しかしこれらは全体形状に対する制約など直感的なインタラクションを阻害する制約を入れることによって解決しようとしており望ましくない。また、これらは3点の確定したノードとの通信をする制約上、通信距離を最短でも3つの機器分取る必要がある。しかしコレクティブなインタフェースを構築するには数多くの計算素子が必要であるため3つ分の長さであっても大勢のノードと通信することになる。このため、さまざまなネットワーク情報を受け取ることになり衝突が起こることや、ノイズが乗ることが予想される。このことから、より近接した少数のノード間だけを対象とするような無線方式が望まれる。

### 3.3.3 グラフ理論における物理配置問題

グラフから物理配置を求める配置問題は古くから VLSI の自動レイアウト問題として関心が高かった。まず配置問題で本論文にかかわる定義、定理を紹介する。

**Definition 3.2** (単位距離グラフ).  $Z^n$  上の部分集合に対応するグラフで、距離が1の点同士が辺としてつながっているグラフのことを単位距離グラフという。

**Definition 3.3** (単位距離配置). あるグラフがあったとき、つながっている辺のみが  $Z^n$  上の部分集合として距離1で配置されるような配置を単位距離配置という。特に断りがない場合は平行移動、回転、反転を同一視する。

**Definition 3.4** (連続グラフ). 任意の点同士が辺をたどって到達可能なグラフのことを連続グラフという。

これらの定義を用いて今回対象とするグラフを定義する。

**Definition 3.5** (形状自在配置、形状自在ネットワーク).  $Z^n$  ( $n = 2, 3$ ) の部分集合  $V$  に対して、連続な単位距離グラフ  $G$  が構成できるとき、 $V$  を  $n$  次元形状自在配置と呼び、 $G$  を  $n$  次元形状自在ネットワークと呼ぶ。

例えば  $n = 2$  の時、部分集合  $V$  としては図 3.5a のようなものがあり、これに対する単位距離グラフ  $G$  は図 3.5b となる。ここで [22] より以下が成り立つ。



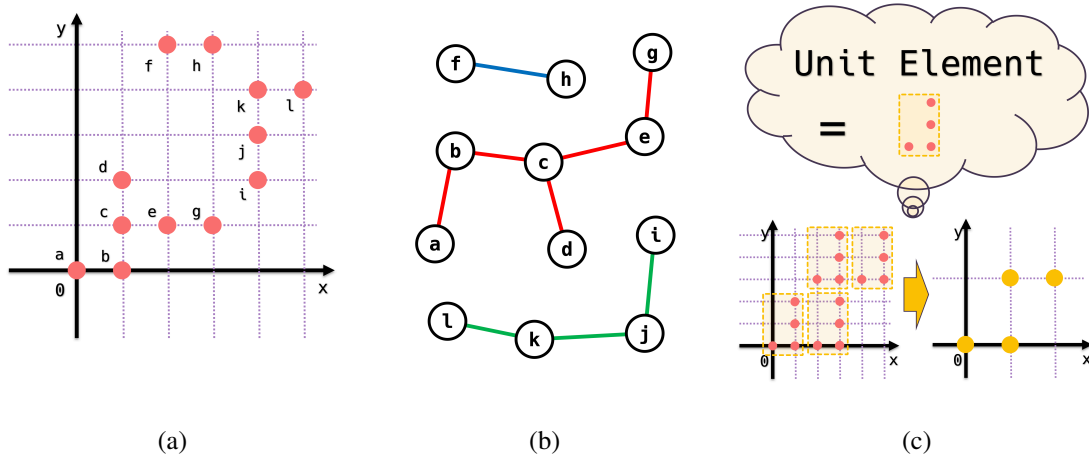


図 3.5: グラフ理論における用語の図的な説明。(b) は (a) から生成された単位距離グラフを表しており、(c) は 3 章で定義される単位素子の配置制約を表している。

**Theorem 3.1.** 子ノードの数が多くても 4 つの木構造グラフに対して二次元上に単位長配置があるかを判定する問題は *NP* 完全である。

2 次元形状自在配置は正方格子上にあり、そのため距離 1 で隣接する点は高々 4 つであるため、単位距離グラフに変換後、木構造グラフとして解釈すると、子ノードの数も高々 4 つである。このことから以下が成り立つ。

**Theorem 3.2.** 2 次元形状自在ネットワークは子ノードが多くても 4 つの木構造グラフに変換できる。

単位長配置が可能なグラフは形状自在ネットワークで表現することができる。仮に形状自在ネットワークの単位長配置問題が *NP* 困難ではないとすると、形状自在ネットワークの単位長配置問題を効率的に解くアルゴリズムが存在することになる。このアルゴリズムを子ノードが多くても 4 つの木構造グラフに変換できる一般のグラフに適応することで、効率的に判定が可能になる。しかし、これは *NP* 困難であるため矛盾が生じる。このことから、2 次元形状自在ネットワークの単位長配置問題は *NP* 困難である。

**Theorem 3.3.** 2 次元形状自在ネットワークの単位長配置問題は *NP* 困難である。

つまり、形状自在ネットワークから形状自在配置を求めることは計算量的観点から困難である。

## 第4章 形状自在計算機ネットワークの問題点と形式化

### 4.1 概要

関連研究であげたネットワークアーキテクチャには問題点が存在する。こうした問題点を評価するために、形式検証を用いてネットワークアーキテクチャの正当性を検証することが有効である。形式検証を行うためには、ネットワークアーキテクチャを数学的に表現する必要がある。

この章では、問題点を解決するために、問題点を定義し、形式検証を行うための数学的な表現を提案する。本論文では代表的なモデル検証ツールである SPIN での利用を念頭に、LTL (Linear Temporal Logic) を用いて問題点を表現する。

### 4.2 各種用語の定義

数理論理学を用いてネットワークアーキテクチャを表現するために、まず各種用語を定義する。

形状自在計算機システムは、多数の素子が集まって構成されるシステムである。この素子の最小単位を定める。

**Definition 4.1** (ノード). ノードとはシステムの構成要素であり、論理的な最小単位である。ノードの集合を  $N$  とする。

形状自在計算機システムのノードを区別するために、ノードには id が割り当てられる。

**Definition 4.2** (id). ノード  $i \in N$  の id とは、ノード  $i$  をネットワーク上で一意に識別するための自然数である。ノード  $i$  の id を  $id_i \in \mathbb{N}$  とする。便宜上 id が定まっていない時は  $id_i = -1$  であるとする。ノード id の集合を  $ID = \{id_i | i \in N\}$  とする。

システムの違いはある時点でのノードの隣接関係と、その隣接関係に基づいて構成されたネットワークグラフの構造によって決まる。これを以下で定義する。

**Definition 4.3** (物理的な隣接関係). ノード  $i \in N$  とノード  $j \in N$  が物理層で直接通信が可能であるとき、 $i$  と  $j$  は物理的に隣接しているという。物理的に隣接関係にあるノード間の関係をリンクと呼び、リンクの集合を  $L \subset N \times N$  とする。

**Definition 4.4** (ネットワークグラフの隣接関係). ノード  $i \in N$  とノード  $j \in N$  がネットワーク層で直接通信が可能であるとき、 $i$  と  $j$  はネットワークグラフで隣接しているという。ネットワークグラフで隣接関係にあるノード間の関係をエッジと呼び、エッジの集合を  $E \subset N \times N$  とする。一般に、 $L \subset E$  である。

隣接関係は変化するため、しばしば時間が重要になる場合がある。時間を強調したい場合は添字  $t$  を用いて、 $N_t$ 、 $L_t$ 、 $E_t$  などと表記する。

これらを用いて、形状自在計算機システムを定義する。

**Definition 4.5** (形状自在計算機システム). 形状自在計算機システムとは、ノードの集合  $N$ 、リンクの集合  $L$ 、エッジの集合  $E$  からなる3つ組  $S = (N, L, E)$  である。

例えば、図 4.1 の左図のような物理配置をもつ計算機の集合があるとする。この物理配置から構成されたネットワークグラフが右図のようになったとする。このとき、

$$N = \{1, 2, 3, 4, 5, 6\}$$

$$L = \{(1, 2), (1, 3), (2, 4), (2, 5), (3, 4), (4, 6), (5, 6)\}$$

$$E = \{(1, 2), (1, 3), (2, 4), (2, 5), (4, 6)\}$$

である。

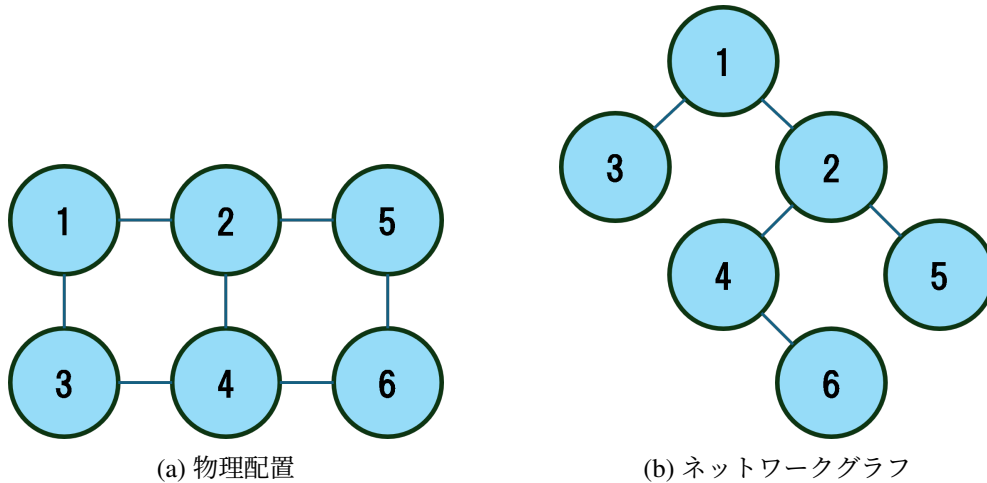


図 4.1: 形状自在計算機システムの例

送信可能なノードすべてが物理的に隣接しているとは限らない。このため、最終的な送信先のノードへ送るための経路を決定する必要がある。

**Definition 4.6** (経路). 経路  $p$  とは、ノード  $i \in N$  からノード  $j \in N$  への通信経路を表す有向パスであり、 $p = (e_0, e_2, \dots, e_{n-1}) \in E^n$  の形である。ここで、 $e_k = \{n_{k_0}, n_{k_1}\}$  とすると、 $n_{k_1} = n_{k+1_0}$ ,  $n_{0_0} = i$ ,  $n_{|p|-1_1} = j$  である。経路の集合を  $P \subset E^n$  とする。

以上で定義した用語は、形状自在計算機システムの実行による理想的な状態を表すものである。次に、ネットワークの構成要素であるノードが保持する情報について定義する。ネットワークプロトコルにバグがある場合、かならずしも理想的な状態を表すものではない。

ネットワークの構成要素であるノードは、通信を行うために隣接ノードの情報を保持している。これを以下で定義する。

**Definition 4.7** (隣接テーブル). ノード  $i$  の隣接テーブルとは、ノード  $i$  が直接通信可能なノード  $id$  の集合  $T_i \subset \mathbb{N}$  である。

各ノードは、通信先のノードに対して物理的な隣接関係があるノードを通じて通信を行う必要がある。このために用いられるのがルーティングテーブルである。

**Definition 4.8** (ルーティングテーブル). ノード  $i \in N$  のルーティングテーブルとは、最終的な通信先のノード  $id$  から候補となる隣接ノード  $id$  を求める関数  $R_i : \mathbb{N} \mapsto 2^{\mathbb{N}}$  である。特に、一通りの通信経路のみが許される場合は、 $R_i : \mathbb{N} \mapsto \mathbb{N}$  と同一視できる。

以下、 $Q, R$  を命題変数とする。

## 4.3 デッドロックの発生

### 4.3.1 問題の概要

デッドロックは、依存関係に循環が生じることで通信ができなくなる状態である。デッドロックが発生すると、通信ができなくなるため、システムが停止する。このため、デッドロックの発生を防ぐことが重要である。

### 4.3.2 問題の定義

**Definition 4.9** (デッドロック). システムの状態列  $S_0, S_1, \dots, S_n$  において、 $S_n$  から遷移できる状態がないとき、デッドロックが発生しているという。

SPIN はラベルによりデッドロックを検出するため、デッドロックの検証条件は不要である。

## 4.4 隣接テーブルの不整合

### 4.4.1 問題の概要

ネットワーク層の隣接テーブルは、通信可能なノードの情報を保持している。この情報が他のノードと不整合を起こすと、通信ができなくなるなどの問題が発生する。このため、隣接テーブルを適切に更新することが重要である。

### 4.4.2 問題の定義

隣接テーブルの不整合を定義する。

**Definition 4.10** (隣接テーブルの不整合). ノード  $i \in N$  の隣接テーブル  $T_i$  が不整合であるとは、以下のいずれかが成り立つことである。

1.  $id_j \in T_i$  かつ  $(i, j) \notin E$  である  $j \in N$  が存在する

2.  $(i, j) \in E$  かつ  $id_j \notin T_i$  である  $j \in N$  が存在する

特に、 $i$  を指定しない場合で隣接テーブルが不整合であると表現したときは、「ある  $i \in N$  について隣接テーブルが不整合である」という論理式を指す。また隣接テーブルが正常であるとは、隣接テーブルが不整合でないことである。

理想は、隣接テーブルが常に正しい情報を保持していることであるが、これは不可能である。このため、ある一定の時間をゆるして隣接テーブルが正しい値に戻ることを検証条件とする。

**Definition 4.11** (隣接テーブルの不整合の検証条件). 隣接テーブルの不整合が発生した場合でも、ある一定の時間内に隣接テーブルが正しい値に戻ることを検証する。

#### 4.4.3 LTL による表現

隣接テーブルの不整合の検証条件を LTL で表現する。 $\Box$  は常に成り立つことを表し、 $\Diamond$  は将来ある時点で成り立つことを表す。

$$\begin{aligned} & \Box(Q \rightarrow \Diamond R) \\ & \text{where} \\ & Q = \text{“隣接テーブルに不整合がある”} \\ & R = \text{“隣接テーブルが正常である”} \end{aligned} \tag{4.1}$$

ここで  $Q = \neg R$  であるため、以下のように変形できる。

$$\Box(\neg R \rightarrow \Diamond R) \tag{4.2}$$

### 4.5 ルーティングテーブルの不整合

#### 4.5.1 問題の概要

ルーティングテーブルは、最終的な通信先のノードから候補となる隣接ノードを求めるために用いられる。隣接ノードが適当な値でない場合、循環パスが発生しデッドロックの発生等、さまざまな問題が発生する。このため、ルーティングテーブルを適切に更新することが重要である。

#### 4.5.2 問題の定義

ルーティングテーブルの不整合を定義する。

**Definition 4.12** (ルーティングテーブルの不整合). ノード  $i \in N$  のルーティングテーブル  $R_i$  が不整合であるとは、以下のいずれかが成り立つことである。

1.  $id_k \in R_i(id_j)$  かつ  $(i, k) \notin E$  である  $j, k \in N$  が存在する

2. ノード  $i$  を終着点以外に含む経路  $p \in P$  が存在し、 $e_{k_0} = i$  とすると、 $id_{e_{k_1}} \notin R_i(id_{e_{k_0}})$  である自然数  $k$  が存在する
3. ある自然数  $n$  であって、任意のノード  $i \in N$  について  $n \neq id_i$  であり、かつ  $n \in R_j(id_k)$  である  $j, k \in N$  が存在する
4. ある自然数  $n$  であって、任意のノード  $i \in N$  について  $n \neq id_i$  であり、かつ  $R_j(n) \neq \emptyset$  である  $j \in N$  が存在する

特に  $i \in N$  を指定しない場合でルーティングテーブルが不整合であると表現したときは、ある  $i \in N$  について不整合であることを指す。またルーティングテーブルが正常であるとは、ルーティングテーブルが不整合でないことである。

最初の条件はルーティングテーブルに登録されているノードは、ネットワークグラフ上で隣接している様なノードであることを示しており、2つ目の条件は、ネットワークグラフ上で表現可能な経路がルーティングテーブルに登録されていることを示している。後半の2つの条件は、不正なノード id がルーティングテーブルに登録されていないことを示している。

隣接テーブル同様、ルーティングテーブルが常に正しい情報を保持していることは不可能である。このため、ある一定の時間をゆるして隣接テーブルが正しい値に戻ることを検証条件とする。

**Definition 4.13** (ルーティングテーブルの不整合の検証条件). ルーティングテーブルの不整合が発生した場合でも、ある一定の時間内にルーティングテーブルが正しい値に戻ることを検証する。

### 4.5.3 LTL による表現

隣接テーブルと同様に、ルーティングテーブルの不整合の検証条件を LTL で表現する。

$$\begin{aligned}
 & \Box(Q \rightarrow \Diamond R) \\
 & \text{where} \\
 & Q = \text{“ルーティングテーブルに不整合がある”} \\
 & R = \text{“ルーティングテーブルが正常である”}
 \end{aligned} \tag{4.3}$$

## 4.6 ノード id に関する問題

### 4.6.1 問題の概要

形状自在計算機システムはネットワーク参加時にノードの id を割り当てる。ノード id はネットワーク内でノードを一意に識別するために用いられるため不整合が生じてはならない。しかし、ネットワーク障害等により正しく id が割り当てられない場合がある。

id の問題は主に以下の2つである。

1. id の重複問題
2. id が連番ではない問題

1 つめは、id が重複してしまう問題である。この問題は、何らかの理由で同一の id が複数のノードに割り当てられてしまうことである。同一ノード id が複数存在すると、ネットワーク内でノードを一意に識別することができなくなるため深刻な問題である。

2 つめは、id が連番ではない問題である。id の最大発行数がルーティングテーブルのエントリ数に関係する。なぜなら、ルーティングテーブルは id を引数としてノードを指定するためである。このため、id の最大発行数が小さければ小さいほどよい。id が連番ではない場合、id の最大発行数を実際のノード数よりも大きくする必要がある。このため、id が連番ではない問題は、ネットワークのスケラビリティに影響を与える。

#### 4.6.2 問題の定義

id の重複問題と id が連番ではない問題を定義する。

**Definition 4.14** (id の重複問題). ある  $i, j \in N$  であって、 $i \neq j$  かつ  $id_i = id_j$  である  $i, j$  が存在するとき、id が重複しているという。

重複とは、同一の id が複数のノードに割り当てられていることである。

**Definition 4.15** (id が連番ではない問題). 現在割り当てられている id の最大値を  $maxId$  とする。すなわち  $maxId = \max_{i \in N} id_i$  である。このとき、ある  $maxId$  未満の自然数  $n$  であって、 $n \notin \{id_1, id_2, \dots, id_{maxId}\}$  である  $n$  が存在するとき、id が連番ではないという。

次に検証条件を考える。ノード id は固定であることを考えると、id の重複問題は常に発生してはならない。

**Definition 4.16** (id の重複問題の検証条件). id の重複問題がいついかなる場合でも発生してはならないことを検証する。

連番問題は、初期化時間のぶれによって発生する可能性があるが、一定時間内に解消され、そして、その後は発生してはならない。

**Definition 4.17** (id が連番ではない問題の検証条件). id が連番ではない問題が発生した場合でも、ある一定の時間内に解消され、その後は発生してはならないことを検証する。

#### 4.6.3 LTL による表現

まず、id の重複問題の検証条件を LTL で表現する。

$$\begin{aligned} & \Box(\neg Q) \\ & \text{where} \\ & Q = \text{“id の重複問題が発生している”} \end{aligned} \tag{4.4}$$

次に、id が連番ではない問題の検証条件を LTL で表現する。

$$\begin{aligned} & \Diamond \Box \neg Q \\ & \text{where} \\ & Q = \text{“id が連番ではない問題が発生している”} \end{aligned} \tag{4.5}$$

「いつか」は初期化完了後であるため以下の様にも表現できる。

$$\begin{aligned} & \neg RU \neg Q \\ & \text{where} \\ & Q = \text{“id が連番ではない問題が発生している”} \\ & R = \text{“初期化が完了している”} \end{aligned} \tag{4.6}$$

## 4.7 同一の情報が複数回処理される問題

### 4.7.1 問題の概要

形状自在計算機システムのリンク層では再送制御が採用されることが多い。この処理に問題があった場合、同一の情報が複数回処理される可能性がある。例えば、id 発行のための情報が複数回送信された場合、システムによっては意図しない id が発行される可能性がある。これは id が連番にならない問題を引き起こす可能性がある。こうした不具合を防ぐために、同一フリットが複数回処理される問題を検証する。

### 4.7.2 問題の定義

フリット全体の集合を  $F$  とする。

**Definition 4.18** (フリットの履歴). ノード  $i \in N$  において、ネットワーク層で処理されたフリットの履歴を  $H_i$  とする。すなわち、 $H_i = (f_1, f_2, \dots, f_n) \in F^n$  である。

**Definition 4.19** (同一の情報が複数回処理される問題). ノード  $i \in N$  にのフリットの履歴  $H_i = (f_1, f_2, \dots, f_n)$  において、ある  $j, k \in \mathbb{N}$  であって、 $j \neq k$  かつ  $f_j = f_k$  である  $j, k$  が存在するとき、ノード  $i$  で同一の情報が複数回処理される問題が発生しているという。特に、 $i$  を指定しない場合で、同一の情報が複数回処理される問題が発生しているとは、「ある  $i \in N$  について同一の情報が複数回処理される問題が発生している」という論理式を指す。

フリットが等価であるとは、基本的にはフリットの id とパケットの id、送信元、送信先が等しいことを意味する。例外として、ハートビートパケットのようにパケットの目的上 id が同一にしても問題ないパケットに関しては、等しくても等価であるとは限らない。このような例外は適宜考慮する必要がある。

どの時点であっても同一の情報が複数回処理される問題は発生してはならないため、以下の検証条件を設定する。

**Definition 4.20** (同一の情報が複数回処理される問題の検証条件). 同一の情報が複数回処理される問題がいついかなる場合でも発生してはならない。



### 4.7.3 LTL による表現

検証条件を LTL で表現すると以下ようになる。

$$\Box(\neg Q)$$

where

(4.7)

$Q$  = “同一の情報が複数回処理される問題が発生している”

## 第5章 形状自在計算機のためのネットワーク インターフェースの提案

### 5.1 概要

形状自在計算機システムのためのネットワークインターフェースの実現について述べる。関連研究で見たように、ネットワークインターフェースの提案は存在するものの、抽象度が高く RTL レベルで実装した例はない。このため、関連研究の提案を実装しようとしたときにはいくつかの問題がある。

この章ではパケット及びフリットの構成を見たのちに、RTL レベルでの実装を見据えて、全体のアルゴリズム、及び、ノード単位のステートマシンレベルでの提案を述べる。なお、全体のアルゴリズムがノード単位である時は、ステートマシンレベルの提案は簡単に実装できるため、省略する。

### 5.2 フリット及びパケットの構成

形状自在計算機システムにおけるパケットは、パケットの構成は、パケットヘッダ、パケットボディ、パケットテールから構成される。パケットヘッダは、最終的な宛先と送信元、長さ、メッセージヘッダを持つ。パケットボディ、パケットテールは、データを持つ。

フリットの構成は、フリットヘッダ、フリットペイロード、チェックサムから構成される。フリットヘッダは、ack かどうか、フリットの種類、ネットワーク層でのチェックを行うかどうか、隣接ノードの宛先と送信元、フリット ID(パケット ID とフリット番号)を持つ。フリットペイロードは、データを持つ。チェックサムは、フリットのエラー検出に用いる。

### 5.3 リンク層

#### 5.3.1 概要

リンク層では信頼性のある通信を実現することが求められる。信頼性向上のためには、データを正しい形で送受信することが求められる。とくに形状自在計算機システムでは、物理層の都合上異なるノードの衝突を検知することが難しい。このため、衝突を回避する仕組みが必要である。

衝突回避には

- 衝突を軽減することによる対策

- 衝突から回復することによる対策
- 衝突を完全に起こさないことによる対策

の3つの方法が考えられる。

WIFI等で用いられる代表的な衝突軽減の方法として、CSMA/CA [23]がある。CSMA/CAは、送信前にチャンネルが空いているかを確認し、空いている場合に送信する方法である。しかしこのアルゴリズムは形状自在計算機システムには適していない。形状自在計算機システムは物理的に隣接したノードの送信のみを検知できるため、

- 検知することができない
- 検知した衝突が正しいとも限らない

の2つの問題点があるためである。

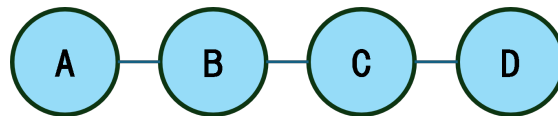


図 5.1: CSMA/CA が適していない例

1つめの問題は、例えば図 5.1 のようなノード配置の時にノード A とノード C が同時に送信を行うと発生する。ノード B では衝突が発生するが、ノード A とノード C では衝突を検知することができない。これは一般に隠れ端末問題として知られているが、形状自在計算機では通常の状態であっても発生する問題である。隠れ端末問題の解決策として、RTS/CTS 処理がある [24]。RTS/CTS 処理はデータの送信前に RTS フレームと CTS フレームによって通信を行う処理のことある。関連のあるネットワーク上のノードに通知し、隠れ端末を回避することができる。しかし、RTS/CTS では通信のオーバーヘッドが大きいという問題がある。

2つめの問題は先の図 5.1 において、ノード B がノード A に向かって送信し、ノード C がノード D に向かって送信する場合に発生する。ノード B とノード C では衝突が発生するが、ノード A とノード D に正しくデータが送信される。

本セクションでは衝突は起こるとした上で回復を行う再送処理アルゴリズムと、座標情報を利用することで衝突を完全に回避する衝突回避処理アルゴリズムについて述べる。

### 5.3.2 構成図

リンク層の構成図を図 5.2 に示す。関連研究からの大きな差分は衝突検知機構の削除である。上記したとおり通常の通信で隠れ端末問題が発生してしまうため、衝突検知機構は意味をなさない。このため、衝突検知機構を削除し、再送処理のみで衝突回避を行う。

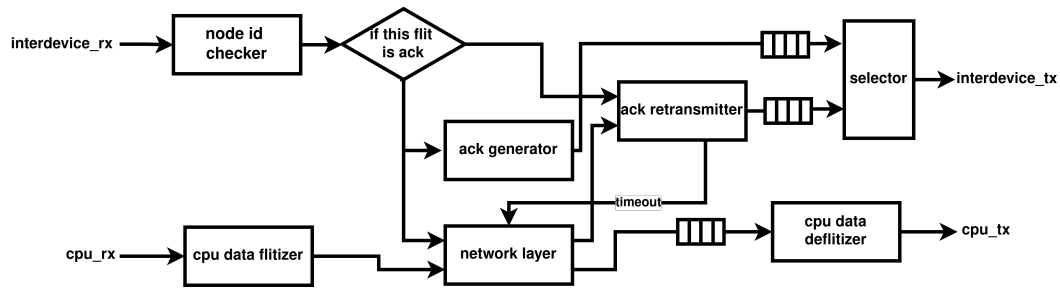


図 5.2: リンク層の構成図

### 5.3.3 再送処理

関連研究で見たように、形状自在計算機システムでは衝突を回避する方法として ack を用いた再送制御が提案されてきた。過去の提案では再送アルゴリズムで衝突を検知した際に、再送を行うとしていた。しかし、これは有効ではない。形状自在計算機では上記の通り隠れ端末問題が発生するため、衝突検知が意味をなさないためである。このため、再送処理は ack をランダム時間待つことによってのみで行う。ランダム時間は関連研究同様に指数バックオフ [4] を用いる。

#### アルゴリズム

---

##### Algorithm 5 再送処理アルゴリズム

---

```

1: // for each flit buffer
2: failedCount ← 0
3: flitBuffer ← currentSendingFlit
4: while True do
5:   // Exponential backoff
6:   waitTime ← baseTime * (1 << failedCount) + random()
7:   sleep(waitTime)
8:   if receiveBuffer is not empty and receiveBuffer's flit is ack then
9:     break
10:  end if
11:  if failedCount = maxFailedCount then
12:    upperUnit ← failedSignal
13:    break
14:  end if
15:  failedCount ← failedCount + 1
16:  SendBuffer ← flitBuffer
17: end while

```

---

#### 5.3.4 衝突回避

座標が完全に分かっている状態での衝突回避アルゴリズムを提案する。なお前提として、座標は格子点上に存在するものとし、隣接したノードのみが通信できるものとする。

##### アルゴリズム (ノード単位)

どの方向からフリットが来たかを判定し、その方向に応じて待ち時間を設定する。同期は隣接しているノードの間でされていれば問題ない。カウンタの初期値は送信周期の2倍とする。送信周期と同様の時間だと、新しく入ったフリットが衝突を引き起こす可能性があるためである。

以下は2次元の場合のアルゴリズムである。3次元の場合は送信周期が7周期となる。

---

**Algorithm 6** 衝突回避アルゴリズム

---

```
1: sendingCycle  $\leftarrow$  5
2: initialCounter  $\leftarrow$  sendingCycle * 2
3: counter  $\leftarrow$  initialCounter
4: while True do
5:   counter  $\leftarrow$  counter - 1
6:   wait(curtain time)
7:   if counter = 0 then
8:     currentNode.send
9:     if collisionDetected then
10:      counter  $\leftarrow$  initialCounter + randomInt()
11:    else
12:      counter  $\leftarrow$  initialCounter
13:    end if
14:    continue
15:  end if
16:  flit  $\leftarrow$  currentSendingFlit
17:  incomingNodePosition  $\leftarrow$  getPosition(flit)
18:  match calculateIncomingDirection(incomingNodePosition, currentNodePosition) do
19:    case Left
20:      counter  $\leftarrow$  1
21:    end case
22:    case Up
23:      counter  $\leftarrow$  2
24:    end case
25:    case Down
26:      counter  $\leftarrow$  3
27:    end case
28:    case Right
29:      counter  $\leftarrow$  4
30:    end case
31:  end match
32: end while
```

---

## 5.4 ネットワーク層

### 5.4.1 概要

形状自在計算機システムでは、ノード間の通信は隣接ノードのみで行う。このため、隣接していないノードとの通信は、隣接ノードを経由して行う。ネットワーク層では、ノード間の通信を行うためのルーティングを行う。

### 5.4.2 構成図

ネットワーク層の構成図を図 5.3 に示す。

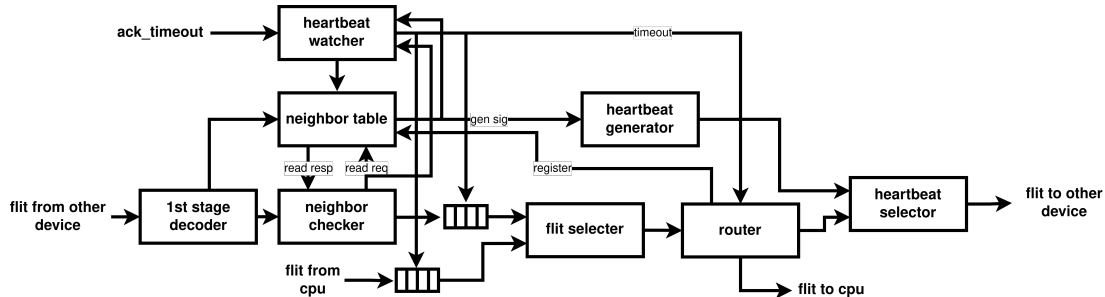


図 5.3: ネットワーク層の構成図

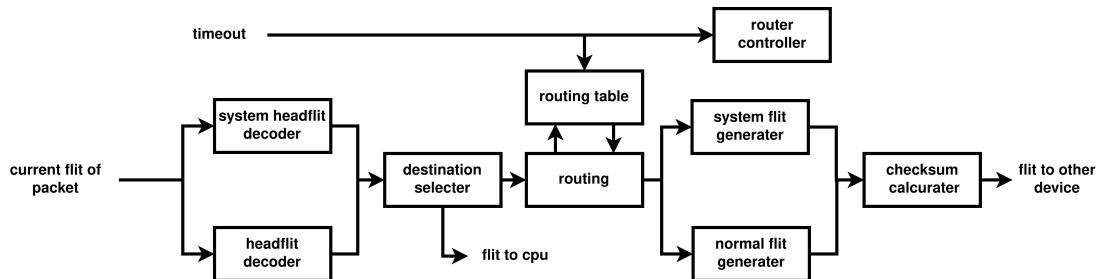


図 5.4: router の詳細

### 5.4.3 ネットワーク参加処理

関連研究との大きな差分は Id 要求 (Id Request, idreq)、Id 承認 (Id Ack, idack) の追加と jack の削除である。idreq はコーディネータに対して自身の Id を要求するフレームであり、idack はコーディネータから Id の割り当てを受けるフレームである。関連研究では jack を用いて Id を割り当て、及びルーティングテーブルの構築を行っていた。しかし、jack の送信中に該当ノードの切断が起きた場合、真ノードを知っている群と知らない群があり、真ノード id を既に登録してしまった切断ノードのエントリを削除することが難しい。仮に一時的に仮 id と真 id の対応を持っておいたとしても、切断を伝えるパケットと jack がこの順でバッファに入っていた場合、仮 id の有効ビットを確認しなければ切断後に jack により登録されてしまうなどの問題が考えられる。

このような煩雑な処理を避けるため、idreq と idack を用いて Id の割り当てを行い、jreq によってルーティングテーブルの構築を行う。jreq によるルーティングテーブルの構築は対象ノードからコーディネータまでに行われるため、パケットが到着順に処理されている場合、安全に切断処理を処理させることができる。またこの方法では仮 id を保存しておく必要がない。

また、その他の差分として、仮 id をルーティングエントリに追加することはない。仮 id のルーティングテーブル登録は、エントリの衝突を引き起こしうる。また仮 id のアドレス

幅が固定されてしまうため、衝突の確率がルーティングテーブルのサイズに依存してしまう。このため、仮 id をルーティングテーブルに登録することは避け、代わりに、仮 id はフリットのペイロード部分に埋め込む。ルートノードからのパケットの経路は親ノードの id を用いて親ノードまで伝送し、そこから random id を用いて伝送する。

## アルゴリズム

---

**Algorithm 7** ネットワーク参加処理

---

```
1: currentNode ← notNetworkJoinNode
2: // pack を受け取るまでは関連研究と同一
3: while not currentNode.receive(pack) do
4:   currentNode.broadcast(preq)
5:   wait for certain time
6: end while
7: currentNode.parent ← pack.src
8: currentNode.send(currentNode.parent, idreq)
9: currentNode.parent.send(root, idreq)
10: // ... normal process
11: root.send(currentNode.parent, idack)
12: // ... normal process
13: currentNode.parent.send(currentNode, idack)
14: currentNode.id ← idack.id
15: currentNode.send(parent, jreq)
16: currentNode.parent.send(root, jreq)
17: currentNode.parent.routingTable.add(currentNode.id, jreq.id)
18: currentNode ← currentNode.parent
19: while currentNode ≠ root do
20:   currentNode.forward(root, jreq)
21:   childNode ← currentNode
22:   currentNode ← currentNode.parent
23:   currentNode.routingTable.add(childNode.id, jreq.id)
24: end while
```

---



## 状態遷移

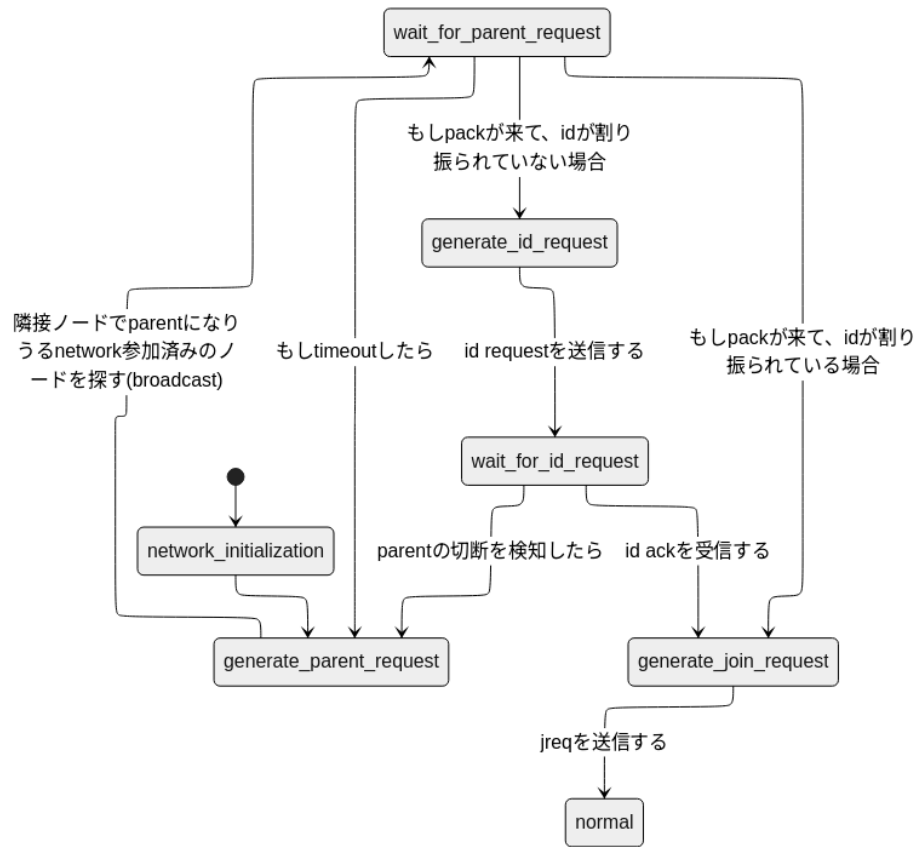


図 5.5: 状態遷移図

### 5.4.4 heartbeat 処理

関連研究では heartbeat 処理についての詳細がなかった。heartbeat 処理は切断状態を管理するのに重要である。ここで、切断状態の管理をリンク層の ack によってのみ更新することはできない。リンク層の ack は物理配置での隣接関係を表すものであり、ネットワーク層の隣接関係を表すものではないからである。このため、heartbeat 処理はネットワーク層の隣接情報をもとに行う必要がある。

heartbeat 処理を行うにあたり、broadcast、unicast のどちらを用いるかが問題となる。以下では broadcast と unicast のそれぞれの実装方法について述べる。

#### broadcast

関連研究では単に broadcast を用いていた。しかし broadcast 処理を単純に行うことでの確認では、ネットワーク全体の状態を把握することができない。broadcast 通信はリンク層での ack 処理を実装しておらず、衝突にたいして脆弱である。また、broadcast 通信をもちいると物理配置で隣接関係にあるノード全体にパケットが送信されるため、ネットワーク

層の隣接関係を伝えることができない。このため、ペイロード部分に隣接ノードの情報を持たせる必要がある。

### unicast

unicast 通信の場合は、リンク層での ack 処理を利用することが可能であるため、broadcast 通信よりも信頼性が高い。unicast 通信の場合、payload 部分に隣接ノードの情報を持たせる必要がない。このため、broadcast 通信よりも一回あたりの通信量が少なくなり、また、受信側の処理が簡単になる。このため固定長のパケットを用いることができる。しかし、broadcast に比べて通信量が増えることや、送信処理が複雑になることなどが問題として挙げられる。

### アルゴリズム (ノード単位)

---

**Algorithm 8** heartbeat 処理 (broadcast)

---

```
1: while network status is active do
2:   wait(curtain time)
3:   neighbor.broadcast(heartbeat)
4: end while
```

---

---

**Algorithm 9** heartbeat 処理 (unicast)

---

```
1: while network status is active do
2:   wait(curtain time)
3:   for all neighbor  $\leftarrow$  neighbors do
4:     if neighbor.sendCounter > maxFailedCount then
5:       continue
6:     else
7:       neighbor.send(neighbor, heartbeat)
8:       neighbor.sendCounter  $\leftarrow$  neighbor.sendCounter + 1
9:     end if
10:    if receiveFlit is from neighbor and it is not ack then
11:      neighbor.sendCounter  $\leftarrow$  0
12:    end if
13:  end for
14: end while
```

---

### 5.4.5 再構成可能な動的再構成ネットワークプロトコル

親ノードの切断を検知した際に、ネットワーク層での隣接情報を完全にリセットし、再度ネットワーク参加処理を行う。シンプルなプロトコルであり動的再構成のベースラインとして用いることができる。

なお親の切断を検知した際はブロードキャスト通信を用いて切断を隣接ノードに通知する。この通知は unicast 通信を用いることもできるが、

- heartbeat 処理がきちんと行われている場合は切断処理が走り必須処理では無い点
- 隣接ノードすべてに通知するため時間がかかる点
- 処理終了までブロッキングされる点

などから採用しなかった。

## アルゴリズム

---

### Algorithm 10 再構成可能な動的再構成ネットワークプロトコル

---

- 1: // 親ノードの切断を検知した場合
  - 2: currentNode.reset(routingTable, neighborTable)
  - 3: currentNode.broadcast(disconnect)
  - 4: currentNode.searchNewParent()
  - 5: currentNode.send(root, jreq)
- 

## 状態遷移

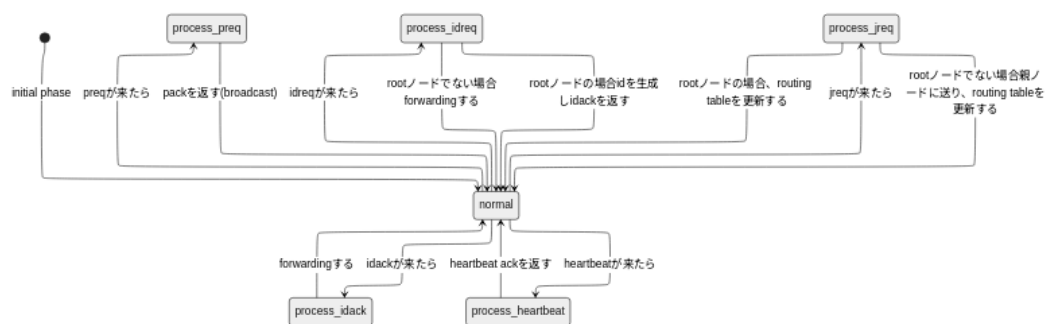


図 5.6: 動的なプロトコル共通な状態遷移図

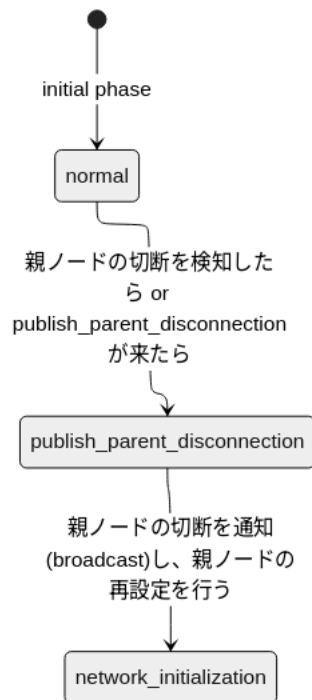


図 5.7: 状態遷移図

#### 5.4.6 木構造を保持した動的再構成ネットワークプロトコル

関連研究であげられているプロトコルでは、不整合が生じる可能性がある。これを再構成フラグフェーズ、探索フェーズ、更新フェーズに分け説明する。

##### 再構成フラグフェーズのアルゴリズム

関連研究では再構成フラグを用いることで、ルーティングの循環を防いでいる。問題はこのフラグをブロードキャスト通信で伝搬させることである。形状自在計算機において、ブロードキャスト通信は信頼性が低い。衝突により、部分木のなかで再構成フラグが正しく伝搬される部分とされない部分ができる可能性がある。このままでは、循環が発生する可能性がある。このため、unicast 通信を用いて再構成フラグを伝搬させる必要がある。再構成フラグを送った後、部分木全体に渡る前に再構成を始めると不整合が生じる可能性がある。このため同期をとり、部分木全体に再構成フラグが伝搬されるまで待つ必要がある。

---

**Algorithm 11** 探索フェーズ

---

```
1: currentNode  $\leftarrow N_{rc}$ 
2: while True do
3:   nextNode  $\leftarrow$  currentNode.nextPreorderTraversal()
4:   if nextNode is null then
5:     if currentNode is  $N_{rc}$  then
6:       currentNode.resetTraversalState()
7:       break
8:     end if
9:     currentNode.send(currentNode.parent, reconstructionFlagComplete)
10:    currentNode  $\leftarrow$  currentNode.parent
11:  else
12:    currentNode.send(nextNode, reconstructionFlag)
13:    currentNode  $\leftarrow$  nextNode
14:  break
15:  end if
16: end while
```

---

**探索フェーズのアルゴリズム**

再構成が伝搬された後、探索フェーズを行う。探索フェーズでは、順に preq を送信し、pack を受信できるかを確認する。pack を受信できた場合、更新フェーズに移行する。

---

**Algorithm 12** 探索フェーズ

---

```
1: currentNode  $\leftarrow N_{rc}$ 
2: while True do
3:   retryCount  $\leftarrow 0$ 
4:   while retryCount < maxRetry do
5:     currentNode.broadcast(preq)
6:     if currentNode gets pack from not  $SubTree(N_{rc})$  then
7:        $N_{nrc} \leftarrow$  currentNode
8:       goto Update Phase
9:     end if
10:    retryCount  $\leftarrow$  retryCount + 1
11:   end while
12:   while True do
13:     nextNode  $\leftarrow$  currentNode.nextPreorderTraversal()
14:     if nextNode is null then
15:       if currentNode is  $N_{rc}$  then
16:         currentNode.resetTraversalState()
17:         break
18:       end if
19:       currentNode  $\leftarrow$  currentNode.parent
20:     else
21:       currentNode  $\leftarrow$  nextNode
22:       break
23:     end if
24:   end while
25: end while
```

---

**更新フェーズのアルゴリズム**

関連研究では更新フェーズで親ノードの更新のみを行っている。しかし、これだけでは不十分である。up\*/down\*ルーティングテーブルは自身の子ノードのみを把握している。このため、子ノードと親ノードの逆転があった場合、もともと子ノードだった側はもともとの親ノードの子ノード全体の情報を持つ必要がある。また、もともと親ノードだった側は、もともとの子ノードを root とする部分木全体を up 方向に変更する必要がある。

---

**Algorithm 13** 更新フェーズ

---

```
1: currentNode  $\leftarrow N_{nrc}$ 
2: while currentNode  $\neq N_{rc}$  do
3:   oldParent.parent  $\leftarrow$  currentNode
4:   currentNode  $\leftarrow$  oldParent
5:   oldParent  $\leftarrow$  currentNode.parent
6: end while
7: currentNode  $\leftarrow N_{nrc}$ 
8: for all childNode  $\in SubTree(N_{nrc})$  do
9:   currentNode.send(childNode, jreqRequest)
10: end for
```

---

状態遷移

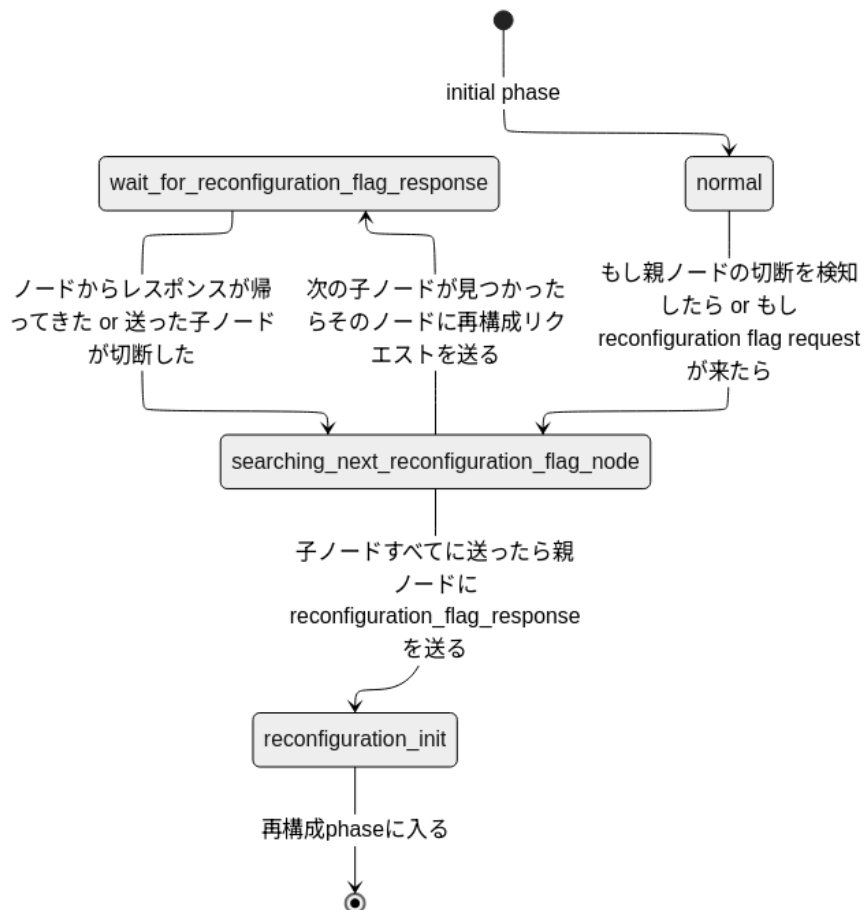


図 5.8: 再構成フラグフェーズ





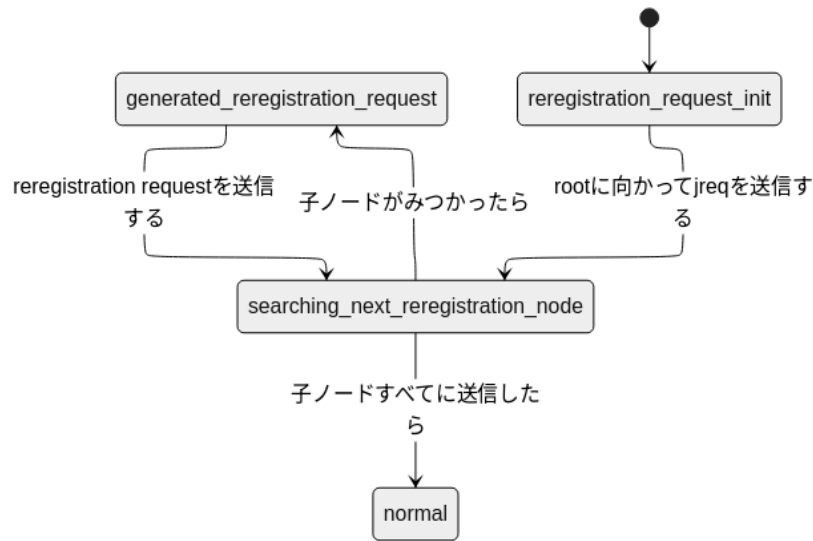


図 5.10: 更新フェーズ

## 5.5 座標推定

### 5.5.1 概要

座標推定アルゴリズムを提案する。関連研究で見たとおり、なにも制約がない場合ネットワークグラフからの座標推定は NP 困難である。しかし、複数素子をあらかじめグループにすることで、非常に簡単に座標推定を行うことができる。

### 5.5.2 アルゴリズム

形状自在計算機では素子が独立している。この独立した素子を複数個まとめて一つの素子として扱うことで座標推定を行う。形状自在計算機の配置を形状自在配置、形状自在配置から生成されたネットワークグラフを形状自在ネットワークと呼ぶ。

今回は、二次元であれば4つをまとめて正方形を、三次元であれば8つをまとめて立方体を一つの素子として扱う。このまとめた素子を単位素子と呼ぶ。この単位素子が格子点上に並んでいる場合を考える。このような場合を規則的な配置と呼ぶ。

座標推定はあるノードを基準とした座標平面を考えるので、基準となるノードを選択する必要がある。これを root ノードと呼ぶ。root ノードは2次元であれば3つ以上必要である。今回は一つの単位素子に属するノードがすべて root ノードであるとし、これを root 単位素子と呼ぶ。

二次元の場合のアルゴリズムを以下に示す。このアルゴリズムは計算量  $O(n)$  である。なお、アルゴリズム中の `localPosition` は、正方形内部の素子の位置を表し、`upperleft`, `upperright`, `lowerleft`, `lowerright` のいずれかである。

`calculateCoordinate` は、受け取った座標情報から自身の座標を計算する関数である。

---

**Algorithm 14** 座標推定

---

```
1: neighborCoordinateList  $\leftarrow$  empty
2: while neighborCoordinateList does not have neighbor's coordinate do
3:   while receiveBuffer is empty do
4:     wait(curtain time)
5:   end while
6:   flit  $\leftarrow$  receiveBuffer.pop()
7:   neighborCoordinateList.add((flit.x, flit.y, flit.candidatePosition))
8: end while
9:  $x_0, y_0, candidate_0, x_1, y_1, candidate_1 \leftarrow$  neighborCoordinateList.filter(neighbor)
10:  $x, y \leftarrow$  calculateCoordinate( $x_0, y_0, candidate_0, x_1, y_1, candidate_1$ )
```

---

### 5.5.3 証明

2次元の場合のみを証明する。形状自在ネットワークはある物理配置から生成されているため、問題となるのは解の一意性と計算量である。解の一意性を単位素子の個数による帰納法で示す。 $n = 0, 1$  の場合は明らかに一意である。 $n = k$  の場合に一意であると仮定し、 $n = k + 1$  の場合にも一意であることを示す。形状自在配置から取り除いた際に、取り除いたあとの部分集合が再び形状自在配置であるような単位素子が存在する。たとえば、生成されうるネットワークグラフの葉になるような単位素子である。この部分集合は仮定により一意である。取り除いた単位素子の追加により一意性が失われることがなければ、 $n = k + 1$  の場合も一意である。取り除いた単位素子の左側に別の単位素子があったとしても一般性を失わない。この別の単位素子との通信により、単位素子が右側にあることがわかる。このようにして、単位素子の個数による帰納法により解の一意性が示された。ここまでの議論のなかで、単位素子の追加にかかる計算量は  $O(1)$  であることがわかるため、全体の計算量は  $O(n)$  である。

## 第6章 形状自在計算機のためのネットワーク インタフェースの検証

### 6.1 概要

本章では、ネットワークインタフェースのモデルを作成し、検証を行う。

ネットワークインタフェースは、形状自在計算機において重要な役割を果たすため正常に動作することが求められる。

### 6.2 検証

#### 6.2.1 検証環境

検証にはモデル検証ツールである Spin を使用する。Spin のバージョンは 6.5.2 を使用する。

今回の検証はモデルが多いため、Spin のビット状態探索を使用する。ビット状態探索は、状態空間をビットベクトルで表現し、状態空間を削減する手法である。この方法を用いることで、現実的な時間とメモリで検証を行うことができる。

Spin 実効時フラグは以下の通りである。

- -a: pan.c に検証コードを生成

pan.c コンパイル時フラグは以下の通りである。

- -DBITSTATE: 網羅探索ではなくビット状態探索を使用

実効時フラグは以下の通りである。

- -m1000000: ビット状態ハッシュ配列に 1000000 メガバイトを使用
- -w27:  $2^{27}$  のハッシュテーブルを使用
- -a: acceptance cycle を検出

#### 6.2.2 検証モデル

検証したモデルはネットワーク参加処理のモデルである。random id は静的に割り当て、同一の id を持つノードは存在しない様にする。なぜなら、random id の衝突が発生すると確実にネットワーク参加処理が失敗するためである。形状は4つのノードからなる直線状のネットワークを用いる。また簡単のため、パケットのロスは考慮せず、位置の変更は行わない。

### 6.2.3 検証条件

検証条件は以下の通りである。

- デッドロックが発生しない
- 親ノードが正しい
- 自ノードの id が定まっている
- 個々のノードが確定していると認識した状態で END ラベルに到達している

すべてのノードが状態遷移を終えた後、すべてのノードが END ラベルに到達していることで、デッドロックの発生なしにネットワーク参加処理が正常に行われたことを示す。

### 6.2.4 検証結果

ビット状態探索を使用して検証を行った結果、エラーは検出されなかった。したがって、このモデルにおいてはネットワーク参加処理が正常に行われることが確認された。

Listing 6.1: 検証結果

```
1 $ ./pan -m1000000 -n -w27 -a
2 ...
3 Depth= 94343 States= 7e+07 Transitions= 2.71e+08 Memory=
   96.176 t= 153 R= 5e+05
4
5 (Spin Version 6.5.2 -- 21 June 2024)
6   + Partial Order Reduction
7
8 Bit statespace search for:
9   never claim + (never_0)
10  assertion violations + (if within scope of claim)
11  acceptance cycles + (fairness disabled)
12  invalid end states - (disabled by never claim)
13
14 State-vector 520 byte, depth reached 94343, errors: 0
15 70963860 states, stored
16 2.0435993e+08 states, matched
17 2.7532379e+08 transitions (= stored+matched)
18 2.7446054e+08 atomic steps
19
20 hash factor: 1.89135 (best if > 100.)
21
22 bits set per state: 3 (-k3)
23
24 Stats on memory usage (in Megabytes):
25 37086.673 equivalent memory usage for states (stored*(State-
   vector + overhead))
26 16.000 memory used for hash array (-w27)
```

```
27 | 7.629 memory used for bit stack
28 | 53.406 memory used for DFS stack (-m1000000)
29 | 19.021 other (proc and chan stacks)
30 | 96.176 total actual memory usage
31 |
32 |
33 |
34 | pan: elapsed time 156 seconds
35 | pan: rate 454546.89 states/second
```

## 第7章 形状自在計算機のためのネットワーク インタフェースの評価

### 7.1 概要

この章では再構成可能な動的再構成可能なネットワークインタフェースを実装し、その性能を評価する。

### 7.2 評価

#### 7.2.1 評価項目

HDL の動作シミュレーションにより形状とネットワークの初期化時間の関係性を評価する。形状には以下のものを使用する。なお横が 1 の場合は直線である。

表 7.1: 動作シミュレーションの形状の一覧

縦	横	対応するノード数
12	1	12
16	1	16
3	2	6
4	2	8
6	2	12
8	2	16
4	4	16
6	6	36

また論理合成の評価では 1 つのネットワークインタフェースの

- 実装面積
- 消費電力
- 最悪遅延 (WNS, Worst Negative Slack)

を評価する。また入力する周波数が 100MHz であることと WNS から、クロック周波数を求める。変更するハードウェアパラメタはネットワークに参加可能な最大のノード数である。ネットワークの最大ノード数はルーティングテーブルのサイズに影響を与える。最大のノード数は以下のものを使用する。なお、ビット幅 14 は合成することができなかった。

表 7.2: ハードウェアパラメタ

最大のノード数	対応するビット幅
4	2
16	4
64	6
256	8
1024	10
4096	12

### 7.2.2 評価環境

HDL の動作シミュレーションには OSS である Verilator(v5.032) を使用する。論理合成の評価環境としては Xilinx 社の Vivado(2024.1) を使用し、Nexys Video ボードをターゲットとする。

### 7.2.3 ネットワーク初期化処理の定性的な評価

ネットワーク初期化処理に関する定性的な理論を述べる。ここでは衝突やパケットロスが完全でないものと仮定する。

簡単のために左端に root ノードがある場合を考える。初期化処理が完了したノードのみがネットワークの参加要求を受け付けることができるため、左側のノードの初期化完了まで右側のノードは初期化処理を開始することができない。また初期化処理には id の割り当てが含まれるが、この割り当ては root ノードとの通信が必要である。このため、左側のノード数に比例して右側のノードの初期化処理にかかる時間が増加する。これらのことを踏まえると、左から  $n$  番目の初期化完了時間を  $T(n)$  とすると、 $T(n)$  は以下のように表される。

$$T(n) = T(n-1) + O(n) \quad (7.1)$$

この式から、 $T(n)$  のオーダーは  $O(n^2)$  であることがわかる。左端に root がない場合は、root の左右のノード数のうち多い方のノード数に比例して初期化処理にかかる時間が増加する。

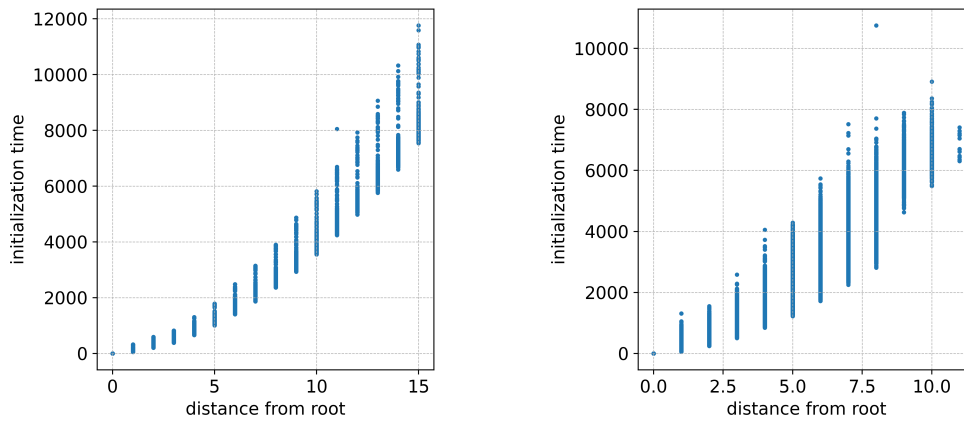
一般の場合は、生成されるネットワークグラフが一意ではない。このような場合でも、あるネットワークグラフが生成されたとした場合は root からのネットワーク距離がもっとも遠いノードの距離に比例して初期化処理にかかる時間が増加すると考えられる。このため、例えば縦  $a$ 、横  $b$  の長方形のネットワークグラフの場合は、最悪でも  $O(a * b)$ 、最良でも  $O(a + b)$  程度の初期化時間がかかると考えられる。平均計算量を求めるには、とりうるグラフ形状を列挙し、最長ネットワーク距離に関する確率分布を求める必要がある。

## 7.2.4 評価結果と考察

### 形状変化によるネットワーク初期化処理の評価

まず直線形状における評価は7.1aのようになった。定性的な評価で考察した通り、直線形状におけるネットワーク初期化処理は $O(n^2)$ であることがわかる。これは直線形上では最大でも前後2つのノードが接続されており、衝突が発生しにくいいため理論値に近い結果になったと考えられる。

次に長方形形状全体における評価は7.1bのようになった。なお、個別の長方形の結果はそれぞれ7.2のようになった。長方形では分散が大きいものの、比較的2次関数に近い時間がかかっていることがわかる。分散が大きい理由としては、衝突による待ち時間が発生しやすいためと考えられる。



(a) 直線形状におけるネットワーク初期化処理の評価 (b) 長方形形状におけるネットワーク初期化処理の評価

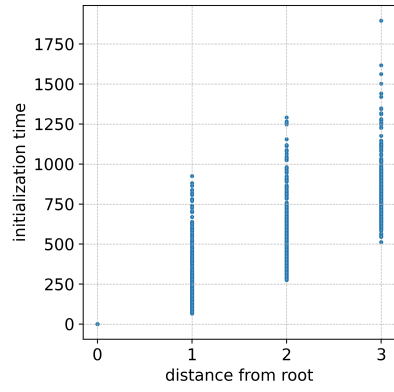
### 論理合成の評価

論理合成では、ノードのビット幅に対する実装面積、消費電力、WNS を評価した。クロック周波数 (MHz) は  $WNS(ns)$  から以下の式を利用して求めた。

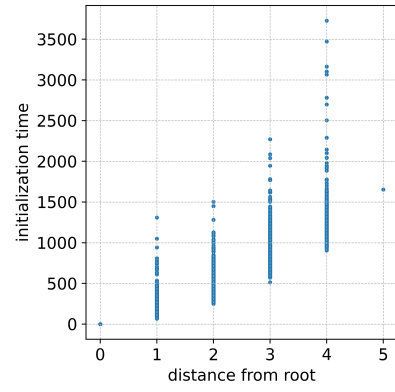
$$f_{clk} = \frac{1}{1/100 - WNS/1000} \quad (7.2)$$

最大のノード数に対する評価結果は7.3のようになった。ノード数に応じて消費電力や実装面積は増加し、クロック周波数やWNSは減少することがわかる。これはルーティングテーブルのサイズやバッファが増加することによるものであると考えられる。ノードのアドレス空間が広がると、その保持に必要なメモリが増えると同時に、キーなどの比較回路も増加する。このため、単純にノード数の2乗に比例して実装面積が増加するわけではなく、より大きな増加率で増加したと考えられる。

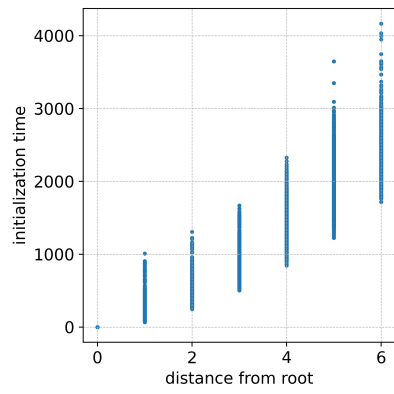




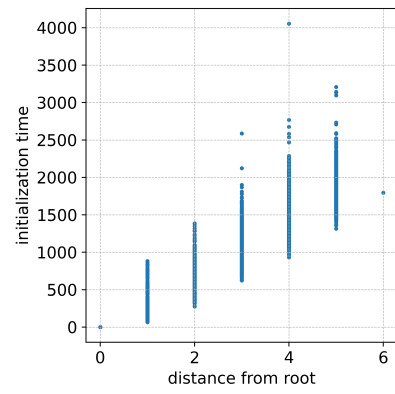
(a) 3x2



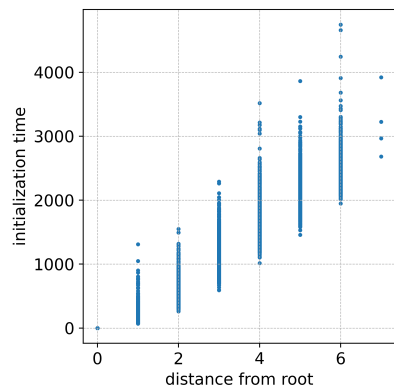
(b) 4x2



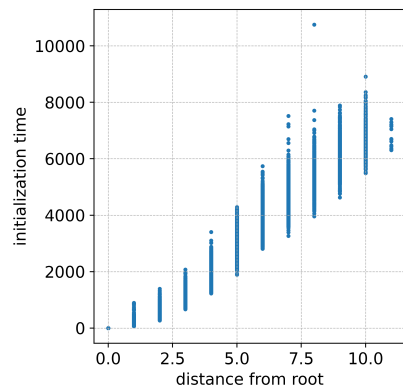
(c) 6x2



(d) 8x2

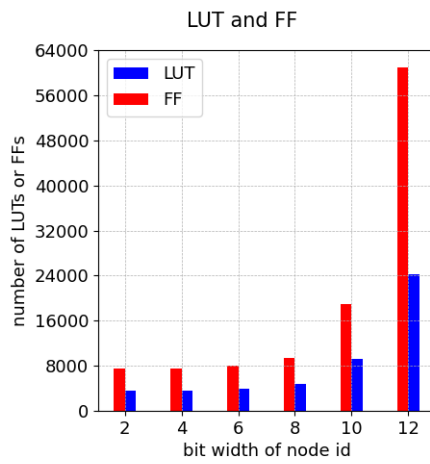


(e) 4x4

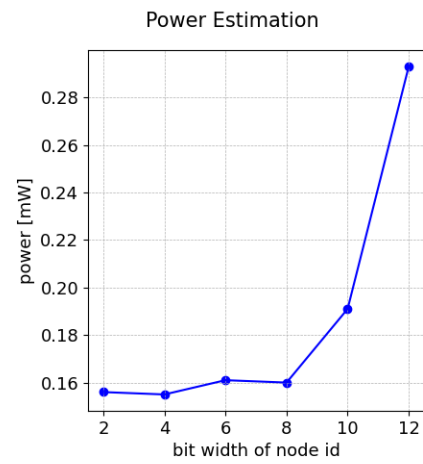


(f) 6x6

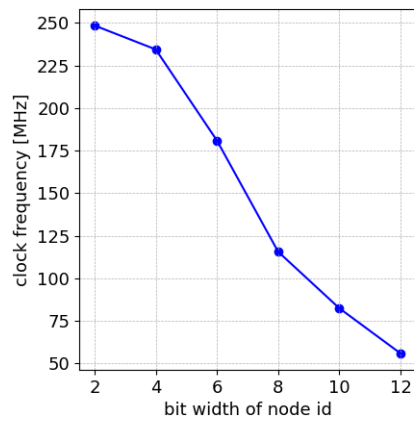
図 7.2: 各長方形形状におけるネットワーク初期化処理の評価



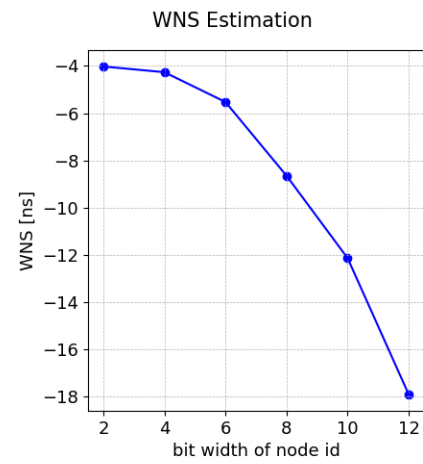
(a) 実装面積



(b) 消費電力



(c) クロック周波数



(d) WNS(Worst Negative Slack)

図 7.3: 論理合成の評価

## 第8章 本論文のまとめと今後の展望

### 8.1 まとめ

本論文では形状自在計算機の実現に向け、ネットワークインターフェースの設計と参加処理の検証を行った。ネットワークインターフェースはソフトウェアシミュレーションを用いて実際の動作を確認し、その性能を評価した。検証ではネットワークプロトコルの問題点を定義し、形式的検証を行った。

### 8.2 今後の展望

これまでは、ハードウェアで実装されたネットワークが存在しなかった。このため、実チップを用いたネットワークの評価が困難であった。本論文により、実チップでの評価が可能になった。このため、CPUとネットワークの統合を行い、形状自在計算機の実現に向けた研究が可能になった。

またベースラインとなるネットワークプロトコルの実装、評価を行った。これにより、より高速なネットワークプロトコルとの比較評価が可能になった。例えば、本論文で示した木構造を保持したネットワークのステートマシンを実際にRTLレベルで実装し、評価を行うことで優劣を比較することができる。

検証においては、ネットワークプロトコルの部分的な形式的検証を行った。形状自在計算機に向けたネットワークプロトコルの検証は初めてである。静的な場合では正しいことが証明された。本論文で示したステートマシンを実装することで切断等を含めたより実践的なネットワークプロトコルの正確性を保証することができる。

## 参考文献

- [1] Ryo Suzuki. Collective shape-changing interfaces. In *Adjunct Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, UIST '19 Adjunct, p. 154–157, New York, NY, USA, 2019. Association for Computing Machinery.
- [2] Junichiro Kadomoto, Takuya Sasatani, Koya Narumi, Naoto Usami, Hidetsugu Irie, Shuichi Sakai, and Yoshihiro Kawahara. Toward wirelessly cooperated shape-changing computing particles. *IEEE Pervasive Computing*, Vol. 20, No. 3, pp. 9–17, 2021.
- [3] Junichiro Kadomoto, Hidetsugu Irie, and Shuichi Sakai. Wixi: An inter-chip wireless bus interface for shape-changeable chiplet-based computers. In *2019 IEEE 37th International Conference on Computer Design (ICCD)*, pp. 100–108, 2019.
- [4] Shun Nagasaki, Junichiro Kadomoto, Hidetsugu Irie, and Shuichi Sakai. Dynamically reconfigurable network protocol for shape-changeable computer system. *IEEE Design & Test*, Vol. 40, No. 6, pp. 18–29, 2023.
- [5] Shun Nagasaki, Junichiro Kadomoto, Hidetsugu Irie, and Shuichi Sakai. Multi-tree network protocol enabling system partitioning for shape-changeable computer system. In *Proceedings of the 21st ACM International Conference on Computing Frontiers*, CF '24, p. 316–317, New York, NY, USA, 2024. Association for Computing Machinery.
- [6] G.J. Holzmann. The model checker spin. *IEEE Transactions on Software Engineering*, Vol. 23, No. 5, pp. 279–295, 1997.
- [7] Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of network-on-chip. *ACM Comput. Surv.*, Vol. 38, No. 1, p. 1–es, jun 2006.
- [8] William James Dally and Brian Patrick Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.
- [9] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. Comput.*, Vol. 36, No. 5, p. 547–553, may 1987.
- [10] Christoph Kern and Mark R. Greenstreet. Formal verification in hardware design: a survey. *ACM Trans. Des. Autom. Electron. Syst.*, Vol. 4, No. 2, p. 123–193, April 1999.
- [11] J Richard Büchi. Symposium on decision problems: On a decision method in restricted second order arithmetic. In *Studies in Logic and the Foundations of Mathematics*, Vol. 44, pp. 1–11. Elsevier, 1966.

- [12] J. Kadomoto, S. Mitsuno, H. Irie, and S. Sakai. An Inductively Coupled Wireless Bus for Chiplet-Based Systems. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 9–10, January 2020.
- [13] *IEEE Standards for Local Area Networks: Carrier Sense Multiple Access With Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*. IEEE Standards 802.3-1985, 1985.
- [14] M.D. Schroeder, A.D. Birrell, M. Burrows, H. Murray, R.M. Needham, T.L. Rodeheffer, E.H. Satterthwaite, and C.P. Thacker. Autonet: a high-speed, self-configuring local area network using point-to-point links. *IEEE Journal on Selected Areas in Communications*, Vol. 9, No. 8, pp. 1318–1335, 1991.
- [15] Per K. Enge. The global positioning system: Signals, measurements, and performance. *International Journal of Wireless Information Networks*, Vol. 1, pp. 83–105, 4 1994.
- [16] Mathieu Le Goc, Lawrence H. Kim, Ali Parsaei, Jean Daniel Fekete, Pierre Dragicevic, and Sean Follmer. Zooids: Building blocks for swarm user interfaces. *UIST 2016 - Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, pp. 97–109, 10 2016.
- [17] Yang Li, John Klingner, and Nikolaus Correll. Distributed camouflage for swarm robotics and smart materials. *Autonomous Robots*, Vol. 42, pp. 1635–1650, 12 2018.
- [18] Nithin Mathews, Anders Lyhne Christensen, Rehan O’Grady, Francesco Mondada, and Marco Dorigo. Mergeable nervous systems for robots. *Nature Communications 2017 8:1*, Vol. 8, pp. 1–7, 9 2017.
- [19] Michael Rubenstein, Christian Ahler, and Radhika Nagpal. Kilobot: A low cost scalable robot system for collective behaviors. *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 3293–3298, 2012.
- [20] Michal Pluhacek, Simon Garnier, Andreagiovanni Reina, M Pluhacek, S Garnier, and A Reina. Decentralised construction of a global coordinate system in a large swarm of minimalistic robots. *arXiv*, 2 2023.
- [21] Mario Coppola, Jian Guo, Eberhard Gill, and Guido C.H.E. de Croon. Provable self-organizing pattern formation by a swarm of robots with limited knowledge. *Swarm Intelligence*, Vol. 13, pp. 59–94, 3 2019.
- [22] Angelo Gregori. Unit-length embedding of binary trees on a square grid. *Information Processing Letters*, Vol. 31, pp. 167–173, 5 1989.
- [23] *IEEE Standard for Information Technology - Telecommunications and information exchange between systems - Local and Metropolitan networks - Specific requirements - Part*

*11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Higher Speed Physical Layer (PHY) Extension in the 2.4 GHz band.* IEEE Standards 802.11b-2000, 2000.

- [24] Kaixin Xu, M. Gerla, and Sang Bae. How effective is the ieee 802.11 rts/cts handshake in ad hoc networks. In *Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE*, Vol. 1, pp. 72–76 vol.1, 2002.

## 著者発表文献

### 国内会議ポスター発表

- 後藤俊樹, 門本淳一郎, 入江英嗣, 坂井修一, ”無線計算機群から構成される形状変化が可能なインタフェースに向けた形状推定アルゴリズム”, インタラクション 2024, 2024 年 2 月, <https://www.interaction-ipsj.org/proceedings/2024/data/pdf/2A-10.pdf>.
- 後藤俊樹, 門本淳一郎, 入江英嗣, 坂井修一, ”形状自在計算機システムに向けたネットワークインタフェース”, DA シンポジウム 2024, 2024 年 8 月.

### 国際会議ポスター発表

- Toshiki Goto, Junichiro Kadomoto, Hidetsugu Irie, and Shuichi Sakai. 2024. Shape Estimation Algorithm for Collective Shape-Changing Interface Using Wirelessly Connected Computers. In Proceedings of the Eighteenth International Conference on Tangible, Embedded, and Embodied Interaction (TEI '24). Association for Computing Machinery, New York, NY, USA, Article 86, 1–7. <https://doi.org/10.1145/3623509.3635267>
- Masato Goto, Ibuki Sugiyama, Kenta Higuchi, Toshiki Goto, Junichiro Kadomoto, Hidetsugu Irie, and Shuichi Sakai. 2024. CommuTiles: Shape-Changeable Modular Computer System Using Proximity Wireless Communication. In Extended Abstracts of the CHI Conference on Human Factors in Computing Systems (CHI EA '24). Association for Computing Machinery, New York, NY, USA, Article 395, 1–5. <https://doi.org/10.1145/3613905.3648669>

### 国内会議登壇発表

- 後藤俊樹, 門本淳一郎, 入江英嗣, 坂井修一, ”形状自在計算機システムに向けたネットワークインタフェース”, DA シンポジウム 2024, 2024 年 8 月.

## 謝辞

本研究を進めるにあたり、多くの方々にご支援いただきました。この場を借りて感謝の意を表します。

指導教員の入江英嗣教授には、研究の方向性や、論文の方向性、また多くの草稿の添削及び校正において大変お世話になりました。

門本淳一郎講師には、形状自在計算機システムに関する深い知見だけでなく、研究の進め方、研究方針の相談や論文の添削などあらゆる面でご支援をいただき大変お世話になりました。

秘書の八木原晴水さん、月村美和さんには出張の際の手続きなどの各種事務手続きおよび備品面確保の面で多くのご支援をいただきました。

研究室の先輩方や同期の方々、後輩たちにはミーティングや発表練習等でのコメントを通じ、多くのことを学ばせていただいたとともに、雑談などを通じて良い刺激を受けることができました。

最後にこれまで温かく見守り、生活面で多くのご支援をいただいた両親に感謝し、謝辞の締めくくりとしたいと思います。