2G－3

超並列ネットワーク・シミュレータ生成系INSPIREにおけるネットワーク記述検証システムの実装*

福士 功治　原田 智紀　朴 泰祐†

筑波大学 電子・情報工学系‡

1. はじめに

超並列計算機における様々な相互結合網を均一の条件下で簡単に評価するため，我々は汎用相互結合網シミュレータ生成系INSPIRE[1]を開発した．INSPIREは，専用言語NDLによるネットワーク諸特性の記述と，C言語によるプロセッサの動作記述から，メッセージ・パッシング型並列計算機のネットワーク・シミュレータを自動生成するシステムである．

NDLは，ネットワークを非常に柔軟に，また比較的容易に記述できるよう工夫されている．しかし，複雑なネットワークを記述する場合にはユーザのミスが生じる可能性があり，現在のINSPIREにおいては，そうしたミスへの対策が十分ではない．

そこで本研究では，INSPIREの支援環境として，NDLによるネットワーク記述を検証するシステムを設計・実装する．本システムでは，NDLプログラムの文法的・意味的検証を行なう他，シミュレーション結果からは得にくいネットワークの静特性の解析も行う．

2.NDLとその検証

NDL[1]はネットワーク諸特性の記述のため，INSPIRE用に開発された言語である．

NDLプログラムはC言語に翻訳されてシステムに組み込まれる．そのため，NDL記述の細部はC言語に依存しており，式やアルゴリズムの記述にはC言語そのものを用いる．また，C言語を用いてユーザ定義変数・関数の宣言などを行なうことができ，それらをNDLプログラム中で自由に用いることができる．このように，NDLはC言語に大きく依存しているため，本検証システムでもINSPIREと同様にNDLプログラムをC言語に翻訳して，コンパイルする手順を採用する．

NDLでは，ネットワーク要素（プロセッサやクロスバ・スイッチ）を宣言し，それらの出力チャネルと入力チャネルを一対ずつ結合してネットワークを構成する．チャネル一つ一つを結ぶことを基本としているため，どのようなトポロジでも記述することが可能になっている．その反面，この柔軟性の高さが，ユーザのミスを誘発する原因となる．そこで，ネットワーク構成の矛盾からこの部分のミスを検出することが，本検証システムの大きな目的の一つである．

また，各ネットワーク要素のルーティング関数（その要素におけるルーティング・アルゴリズム）はメッセージを基準に記述される．つまり，あるネットワーク要素に到着したメッセージが次に獲得する出力チャネル（次にどの要素に進むか）の選択を記述する．また，メッセージの属性やネットワークの混雑状況を調べるシステム関数も用意されており，これを用いて様々なルーティング・アルゴリズムを記述することができる．

3. 検証システムの仕様

3.1 対象とするネットワーク

本システムで検証の対象とする記述は，

1) 全ノード対全ノードの通信が可能で，
2) ルーティング決定はシステム関数にのみ依存し，
3) ルーティング決定はメッセージの待ち時間に依存しない．

ネットワーク記述でなければならない．

INSPIRE及びNDLが対象とする並列計算機において，全ノード間でのデータ交換が不可能であるものは極めて少数であると考えられ，1)の制約は特に問題にはならない．

2)の制約は，システム関数の値，つまり，ルーティング対象のメッセージ自身の属性とネットワークの状態が等しい限り，ルーティング関数は同一の振舞いをする必要があり，他の要因によってその振舞いが変化してはいけないことを示している．ただし3)の制約により，メッセージの待ち時間を得るシステム関数の利用はできない．これらの制約は時刻・時間経過・乱数などを利用するルーティングを考える際には問題となる．しかし，そのような要因の影響は，実際のシミュレーションを行なって調べることがふさわしいと考えられる．よって，シミュレータ生成以前に使用されるべき本システムの対象とはしない．

3.2 検証内容の概要

本システムでは大きく分けて

1) 文法的な検証
2) 意味内容の検証
3) ネットワークの静特性の解析

の，三つを行なう．

文法的な検証では，INSPIREによってシミュレータが生成できないような記述をチェックすることができる．意味内容の検証では，正常動作しないシミュレータが生成されてしまうような記述をチェックし，問題箇所や原因を特定する．そして，ネットワークの静特性の解析では，シミュレータとは異なる側面からネットワーク特性を解析する．

3.3 文法的な検証

文法的な検証では，与えられたNDLプログラムの構文を解析し，NDLの構文に沿わない場合文法エラーとして報告する．これと同時に，変数名の衝突，型の不一致などのエラーを検出，報告する．

ただし本システムでは，NDL特有の構文についてのみチェックし，C言語による記述の部分はチェックしない．（C言語部分の文法は，後にCコンパイラによってチェックされる．）

3.4 意味内容の検証

意味内容の検証では，まず各種宣言について，宣言の不足や二重宣言，不正なパラメータが与えられていないかをチェックする．

宣言などが妥当であれば，それにしたがってネットワークを構成し，そのネットワークについて，

1) 未接続のチャネルがないか（警告のみ）
2) 各チャネルが一対一で接続されているか
3) 全ノード対全ノード通信を行なえる経路が存在するか

を検証する．

ネットワークが妥当であれば，各ルーティング関数を評価し，

1) route命令やシステム関数に与えられた引数が適

---

*Network description verification system for massively parallel network simulator generation system INSPIRE

†Koji Fukushi, Tomoki Harada, and Taisuke Boku

‡Institute of Information Sciences and Electronics, University of Tsukuba

正か
2) 全ノード対全ノード通信を行なえるルーティングが存在するか

を検証する. 適応ルーティングを行なうアルゴリズムを記述をした場合でも, すべての適応の可能性を尽くして検証を行なう.

3.5 ネットワーク静特性の解析

ネットワーク静特性の解析では, まず平均及び最大ネットワーク距離を測定する. ここで, ネットワーク距離とはノード間の最短経路長を指し, これはネットワーク・トポロジのみに依存する値である.

次に, 平均及び最大ルーティング距離を測定する. ルーティング距離とは, 実際のルーティングの結果としてメッセージが通過する経路長を指す. また適応ルーティングの場合には, 平均及び最大の適応度を調べる。適応度とは, 一組の送受信ノード間に存在し得る経路の総数をいう.

最後に, チャネル依存グラフにサイクルが存在するかを調べる. チャネル依存グラフとは, メッセージがチャネルを使う順序, あるいは, メッセージがネットワーク中をどのように進むかを有向グラフとして表したものである. チャネル依存グラフにサイクルが存在しないことは, 一般にネットワークがデッドロック・フリーであることの十分条件となり, 特に, 固定ルーティングにおいては必要十分条件となることが知られている [2].

4. 検証システムの構成

NDL では, 式やアルゴリズムは C 言語を用いて自由に記述することができ, C コンパイラによるコンパイルをせずにその内容を解析するのは困難であると考えられる. そこで, 本システムの実行は,

1) NDL プログラムを C 言語へ変換
2) 1) を検証システム本体と共にコンパイル
3) 2) の結果生成された検証ツールを実行

の三段階に分けて行なわれる.

NDL プログラムの文法的な検証は 1) の変換を行なうトランスレータで, 意味内容の検証とネットワーク静特性の解析は 3) の検証ツールで行なう.

以上のようなシステムを, yacc, lex, C 言語を用いて作成した.

4.1 トランスレータ

トランスレータは, NDL プログラムを解析し C 言語のネットワーク記述に変換し, また変換不能な構文が与えられると文法エラーとして問題箇所を指摘する. さらに, NDL 特有の変数について, 型の不一致や二重宣言などのエラーを検出・報告する.

NDL プログラム内の C 言語による記述は, 変換後の C 言語ネットワーク記述にそのまま複写され, トランスレータでのチェック対象とはならない. ただし, C コンパイラによるエラー報告でも, 変換元の NDL プログラムでの問題箇所を指摘してくれる.

4.2 検証ツール

検証ツールはまず, ネットワーク記述のエラーを検出しながら, その記述に従ったネットワークをシステム内部に構成する. そして, 構成されたネットワークについて, 全ノード間の経路確認をし, ネットワーク距離を求める.

ルーティングの検証では, 全ノード対全ノードの転送をシミュレートして実際の転送パターンに即してルーティング関数を評価・検証する. この際, ルーティング関数から呼ばれたシステム関数の返り値を制御してあらゆるネットワーク状態を疑似的に再現し, 適応ルーティングの適応の可能性を尽くす. この転送シミュレーションの履歴から, ルーティング距離と適応度の測定, さらにチャネル依存グラフの作成が行なわれる.

チャネル依存グラフのサイクル検出では, 多進木と同様にしてグラフを探索する. 探索途中で, あるチャネルの枝の中に自分の根となっているチャネルのどれかが存在すれば, サイクルが存在することになる.

表 1: 8 ノード単方向リング解析例

	修正前	修正後
平均ネットワーク距離	4	4
最大ネットワーク距離	7	7
平均ルーティング距離	4	4
最大ルーティング距離	7	7
CDG のサイクル	exist	none

5. 検証システムの評価

本システムを利用することで, シミュレーション結果からは分かりにくかったデッドロックの可能性を知ることもできる. 以下, その簡単な例を示す.

ここで評価用に用いたネットワーク記述は, 8 ノードの単方向リングの記述である. ルーティング・アルゴリズムは, デッドロックを起こすもの (修正前) と, 同記述をデッドロック・フリーとなるように修正したもの (修正後) の二つを用いた.

デッドロックを起こすルーティング・アルゴリズムでは, 各ノードはメッセージを順次隣のノードへ転送するだけである. 修正したアルゴリズムでは, メッセージの宛先アドレスとメッセージヘッダのアドレスとの大小によりバーチャル・チャネルを使い分け, デッドロックを回避する.

両記述の検証ツールによる解析結果は表 1のようになった. 両記述ともネットワーク距離, ルーティング距離は同じだが, 修正後はチャネル依存グラフ (CDG) のサイクルが解消され, デッドロック・フリーになったことが確認できた.

4×4×8 の三次元ハイパクロスバ網に対し, 固定ルーティングと, 最短経路を通る適応ルーティングの双方について実行時間を調べた。SUN の SS-5/75 上で, トランスレータはどちらも 1.4 秒, 検証ツールは前者は 2.2 秒, 後者については 41.8 秒を費やした。

6. おわりに

本研究では, 汎用相互結合網シミュレータ生成系 INSPIRE の支援環境として, NDL プログラム検証システムを設計・実装した. また, 本システムを用いることで, NDL プログラムのデバッグが容易になることが期待でき, さらにシミュレーションからは得られないネットワーク特性を得られることを確認した.

現在の検証システムでは, 文法エラーは発生箇所を指摘するに留まっているが, エラー原因の特定を行なうことも必要であると考える. また, チャネル依存グラフの解析から, サイクルの有無以外の情報を抽出することも検討中である.

謝辞

本研究に関し貴重な御意見を頂いた, 筑波大学西川博昭助教授ならびに坂井修一助教授, アーキテクチャ研究室諸氏に深く感謝します. なお, 本研究の一部は創成的基礎研究費 (08P0401) の補助によるものである.

参考文献

[1] 原田智紀 他, “並列処理ネットワークのための性能評価用シミュレータ生成系 INSPIRE” 情処研報 Vol.95, No.80, pp.65-72

[2] W.J.Dally, and C.L.Seitz, “Deadlock-Free Message Routing in Multiprocessor Interconnection Networks” IEEE Transaction on Computers, Vol.C-36, No.5, May 1987, pp.547-553