計算機アーキテクチャ 113－9
（1995. 8. 24）

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

原田 智紀　曽根 猛　朴 泰祐　中村 宏　中澤 喜三郎

筑波大学 電子・情報工学系
〒305 つくば市天王台 1-1-1

Tel: 0298-53-6912　Fax: 0298-53-5206
{tomoki,sone,taisuke,nakamura,nakazawa}@arch.is.tsukuba.ac.jp

あらまし

本稿では、汎用ネットワーク・シミュレータ生成系INSPIREを提案する。本システムでは、ユーザがネットワーク記述言語NDLによりネットワークの資源、ネットワークのトポロジ、ルーティング・アルゴリズムといったネットワークの諸特性を記述できる。また、ユーザは各PUの動作をC言語により記述することができる。INSPIREではこれらの機能により、あらゆる転送パターンにおける種々のネットワークの性能評価を行なうことができる。INSPIREはyacc、lexによるNDLトランスレータと、C言語によるシミュレーション実行用のカーネルライブラリからなり、現在各種UNIXワークステーション上で稼働中である。

INSPIREのこれらの特性を活かし、ネットワーク間の転送性能比較を行なった。その結果、NDLが様々な直接網・間接網の記述を行なえること、それらのネットワーク上での性能評価を容易に行なえることを確認した。また、生成されたシミュレータの実行速度は、特定のネットワーク専用に記述されたシミュレータのものと比べ約40%程度の低下が見られた。しかし、これはINSPIREの汎用性を考えるとほぼ満足できるものと考えられる。

A Network Performance Evaluation Simulator Generation System INSPIRE for Massively Parallel Processing

Tomoki HARADA　Takeshi SONE　Taisuke BOKU
Hiroshi NAKAMURA　Kisaburo NAKAZAWA

Institute of Information Sciences and Electronics
University of Tsukuba
1-1 1 Tennodai, Tsukuba 305

Tel: 0298-53-6912　Fax: 0298-53-5206
{tomoki,sone,taisuke,nakamura,nakazawa}@arch.is.tsukuba.ac.jp

Abstract

In this paper, we propose general purpose network simulator generation system named INSPIRE. In this system, a user can describe various characteristics of the network, for instance, network resources, network topology and routing algorithm, in a network description language called NDL. A user also can describe the behavior of PUs (Processing Units) in C language. With these features of INSPIRE, we can evaluate performance of various networks and various patterns of message transfer. INSPIRE consists of an NDL translator written in yacc and lex, and a simulation kernel library written in C. INSPIRE is now available on several UNIX workstations.

Using these characteristics of INSPIRE, we compared message transfer performance on several networks. As a result, we confirmed that we can describe various kinds of networks by NDL, and easily evaluate performance of message transfer on them. We also compared the speed of generated simulators by INSPIRE with specially dedicated simulators. In the results, the speed of INSPIRE simulators are degraded about 40% from the native simulators. While it is caused by the generality of INSPIRE, its performance is enough for various network simulations.

1 はじめに

現在、高性能を目指した数千台規模のPU (Processing Unit) を持つ超並列計算機システムが、研究あるいは実装されている。このような大規模な計算機では、ネットワークにおける高い転送性能が要求される。

現在の超並列計算機の多くは直接網を用いている。直接網ではPU間をスイッチを介すことなく直接リンクにより結合している。直接網の典型例としてメッシュ/トーラス (AP1000[1], T3D, Paragon)、ハイパキューブ (nCube)、ツリーなどが挙げられる。これらのネットワークはPU間の結合にスイッチを必要としないため、PU間をスイッチを介して結合する間接網に比べ低コストで実装できる。その反面、ネットワークの直径・PU間の平均距離が比較的大きくなるという欠点を持つ。特にランダム転送を行なった場合に、ネットワークの直径・PU間の平均距離がネットワークの転送性能の低下に大きな影響を与える。

一方、間接網では直接網と異なりPU間の接続をスイッチを介して行なう。このクラスの代表はMIN (Multistage Interconnection Network) である。MINではPU間の結合を多段に接続した小規模クロスバ・スイッチを介して行ない、PU間の距離が一定であるという特性を持つ。従って、ランダムな転送パターンに対しては効果的であるが、近接転送のような局所性のある転送には不向きである。また、このクラスのネットワークは一般に2入力2出力程度の小規模のスイッチを用いているため、数千台規模のPUをもつ超並列計算機システムではPU間の距離が比較的大きくなってしまう。現在では、VLSI技術の発展に伴い中規模のクロスバ・スイッチが比較的容易かつ低コストに実現できるようになり、このような中規模のサイズのクロスバを用いた様々なネットワークが実装されるようになった。例えば、ADENART[2] で使用されるハイパクロス・ネットワーク、CM-5で用いられているファットツリー・ネットワーク、CP-PACS[3] で使われているハイパクロスバ・ネットワークなどである。

直接網や間接網にはまだ研究の余地があり、ネットワークの性能評価の必要性は高い。ネットワークの評価は計算機シミュレーションにより行なわれることが多く、シミュレーションにより得られたデータは新しい超並列計算機システムを開発する上で非常に有効である。ところが現在のシミュレータのほとんどは特定のネットワークを対象として個別に作成されており、種々のネットワークを同一の条件の下で評価することができない。そのため様々なネットワークを同じ条件の下で評価できる汎用シミュレータが必要である。

このような汎用シミュレータの1つとしてINSIGHT [4] がある。INSIGHTでは幾つかのPU間の接続を表す命令を組み合わせることにより、対象とするネットワークのトポロジを記述することができ、これにより様々なネットワークの評価を行なうことができる。しかしながら、INSIGHTは直接網をのみを対象としており、間接網の評価を行なうことができない。したがって直接網・間接網の両方を扱える、より柔軟な汎用シミュレータが求められる。

そこで本稿では、汎用ネットワーク・シミュレータ生成系INSPIRE (Interconnection Network Simulator with Programmable Interaction and Routing for performance Evaluation) を提案する。このシステムでは、ユーザがネットワークの諸特性、PUの動作を各々独立して記述でき、これらの記述に従うシミュレータを生成することができる。ネットワークの諸特性はNDL (Network Description Language) と呼ばれる専用言語により記述され、PUの動作はC言語により記述される。NDLはトランスレータによりC言語に変換され、これとPUの動作を記述したファイルおよびカーネルを結合することによりシミュレータが生成される。

以下本稿ではINSPIRE、特にNDLの概要とその記述力を紹介し、幾つかのネットワークに対しINSPIREによって生成されたシミュレータを用いた評価を行なう。これらにより、INSPIREの有効性を確認する。

2 INSPIREの概要

本節ではINSPIREの概要について述べる。INSPIREの構成図を図1に示す。INSPIREはカーネルおよびトランスレータにより構成され、NDF (Network Description File) とPBF (PU Behavior File) の2つのファイルを入力とし、これらを用いて1つのシミュレータを生成する。

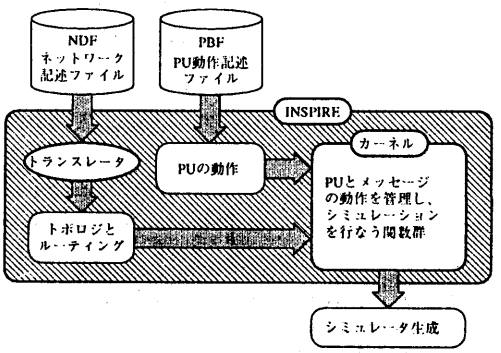

図1: INSPIREの概要図

2.1 NDF

NDFとはネットワークの規模、ネットワークの資源、ネットワーク・トポロジ、ルーティング・アルゴリズムといったネットワークの特性を記述するファイルのことであり、ネットワーク記述専用言語NDLにより記述される。NDLについては3節で詳しく述べる。

2.2 PBF

PBF はPU の動作を記述するファイルであり、C 言語により記述される。PBF には各PU の内部処理、PU 間通信のための送受信動作、通信の際のメッセージの属性 (長さ、データ、タイプ) を記述することができる。ユーザはこのようにPU の動作を自由に手続き的に記述することができ、簡単なパターンの転送から実アプリケーションに基づくような複雑なパターンの転送まで、あらゆるパターンの転送を実現することができる。

ユーザはPBF 内に以下の2 つの関数を定義しなければならない。

- pu_init(): この関数はPU の初期状態を記述する関数であり、各PU 毎に呼び出される。各PU で幾つかの変数を必要とする場合には、C 言語の関数 malloc() により必要とする変数を確保することができる。確保した変数はカーネルにより管理され必要に応じて関数pu_exec() で参照される。また、実行時にコマンドラインから引き渡されるユーザ定義の変数はこの関数内で認識される。

- pu_exec(): この関数はPU の動作を記述する関数であり、次のPU 動作を決定する際にカーネルにより呼び出される。この関数内ではpu_init() で確保した変数を使用することができる。また、この関数は返り値により各PU の状態を決定する、カーネルはこの返り値を受けて各PU の動作をシミュレートする。

関数pu_exec() はPU の動作を決定するためにフェーズ分けを行なう。例えば、処理と処理の間にメッセージの送信を行なう場合を考える。まず処理のフェーズを実行し、次にメッセージの送信のフェーズを実行する。最後に処理のフェーズを実行するという流れになる。フェーズを管理するにはpu_init() で確保した変数を用いる。pu_exec() はこの変数の値を認識することにより、どのフェーズを実行するべきかを判断する。

このようにpu_exec() ではフェーズを分けてPU の動作を決定する。このため、プログラミングは若干難しいが、あらゆる転送パターンを実現できる。

2.3 システム

INSPIREはカーネルおよびトランスレータにより構成される。

- トランスレータ: yacc と lex により記述されている。トランスレータはNDL プログラムの構文を解析し、C 言語プログラムへの変換を行なう。

- カーネル: PU とメッセージの動作を管理しシミュレートする。また、カーネルはシミュレーションの終了時に自動的に統計情報を表示するという機能を持つ。カーネルのソースはC 言語により記述され、最終的なシミュレーション生成に便利なようにライブラリ化されている。

カーネルはシミュレーション実行時に以下のパラメータを必要とする。

- リンクのバンド幅: ネットワーク中のすべてのリンクのバンド幅を表す。PBF ではメッセージ長をbyte 単位で表すため、リンクのバンド幅はbyte/clock として表される。

- シミュレーション・クロック: シミュレーションの総実行クロック数を表す。特に指定すればシミュレーション時間無制限も指定でき、この場合はすべてのPU の処理が終了するまでシミュレーションを継続する。

現在INSPIRE では、メッセージの転送方式としてwormhole 方式のみをサポートしており、各チャネルのバッファサイズは1 としている。転送されたメッセージはネットワーク中のこのバッファを獲得し、その間、他のメッセージはそのバッファを利用できなくなる。このようにしてINSPIRE ではメッセージ同士の衝突をシミュレートする。

INSPIRE では、様々な粒度のメッセージ転送を扱うことを前提とする。このためシステムの状態がほぼ毎時刻変化することを想定し、タイム・スライス方式の時刻管理を採用している。INSPIRE では毎時刻PU およびメッセージの状態遷移を確認し、それに応じた動作をシミュレートする。

3 NDL の概要

本節ではNDL (Network Description Language) の仕様について簡単に述べる。NDL はネットワークの諸特性を記述するためにINSPIRE 用に開発された言語である。NDL の記述は関数定義部、資源記述部、結合記述部、ルーティング・アルゴリズム記述部に分かれる。

関数定義部では関数net_config() をC 言語により定義する。この関数はシミュレーション実行時にコマンドラインから引き渡される引数を認識し、ネットワークの形状を決定するというものである。例えば、ハイパキューブは次元数、2 次元トーラスは各次元のサイズを、関数net_config() 内でシミュレーション実行時に決定することができる。また、関数定義部ではユーザがC 言語の変数および関数を定義することができ、このユーザ定義の変数および関数を用いることによって、より柔軟な記述を行なうことができる。

資源記述部ではネットワーク資源の宣言を行なう。ユーザはPU およびスイッチを多次元に宣言できる。また、各ネットワーク資源の入力チャネルと出力チャネルのサイズについても宣言を行なう。特に、チャネルのバーチャル化を行ないたい場合は、バーチャル化の宣言を行なうこともできる。

結合記述部では資源記述部で宣言したネットワーク資源間の結合を記述する。この記述をすべてのネットワーク資源に対して行うことにより、ネットワーク・トポロジを表現する。

ルーティング・アルゴリズム記述部では各ネットワーク資源におけるルーティング・アルゴリズムを記述する。メッセージはこのルーティング・アルゴリズムを用いて次に行くべき出力先を決定する。

3.1 変数

超並列計算機では非常に多くのPUおよびスイッチが必要とされる。従って、これらを表現するためにはFortranのDO文、C言語のfor文のようなループ構造が必要となる。NDLではこれを簡単に示すために、以下の2つの特別な変数を用意している。これらの変数はそれそのものがループ構造と等価の働きをする。

- 範囲変数: ある範囲に含まれる整数の集合を表す。範囲変数は以下のように宣言される。

  range 範囲変数名 = [下限値 : 上限値], … ;

  このようにして宣言された範囲変数は下限値から上限値までの範囲の整数の集合を表し、次に示す範囲付変数の宣言に用いられる。

- 範囲付変数: ある範囲中の下限から上限までのすべての整数値が適用される得る整数型変数である。範囲付変数は以下のように宣言される。

  var 範囲付変数 **in** [下限値 : 上限値], … ;

  また、範囲変数を用いて次のように宣言することもできる。

  var 範囲付変数 **in** 範囲変数, … ;

このようにして宣言された範囲付変数は複数の式を1つの式に包括するために使用される。例えば、a(1),a(2),…,a(n)は下限が1、上限がnの範囲付変数iを用いるとa(i)というように簡略化できる。また、これはa(i,j)のように多次元化して用いることもできる。これらの変数の導入により、ループ構造を明示することが不要になる。

3.2 資源記述部

資源記述部では、ノード、スイッチといったネットワーク中の資源の宣言を以下の形式で記述する。

```
resource{
    node ノード変数, … ;
    switch スイッチ変数, … ;
    switch_size
      ノード変数 {i(入力数),o(出力数)}, … ,
      スイッチ変数 {i(入力数),o(出力数)}, … ;
}
```

資源記述部はノード宣言部、スイッチ宣言部、スイッチ・サイズ宣言部に分かれる。

node ではじまる文がノード変数宣言部であり、**switch** ではじまる文がスイッチ変数宣言部である。ノード変数とスイッチ変数はそれぞれPUとスイッチを表す変数であり、これらは多次元配列として宣言することができる。ノード変数とスイッチ変数の宣言部は同じ形式である。ノード変数は必ず必要であるが、スイッチ変数にはその制約はない。例えば、直接網にはネットワーク中にスイッチがないため、直接網を記述する場合にはスイッチ変数を宣言する必要はない。

すべてのノード変数とスイッチ変数に対して、入力チャネルと出力チャネルの数を宣言する必要がある。この宣言部はスイッチ・サイズ宣言部と呼ばれ、**switch_size** ではじまる文がそれである。スイッチ・サイズ宣言部の **i** と **o** はそれぞれ入力と出力を表し、それらの予約語に続く括弧内の数値が入力および出力チャネル数を表す。また、スイッチ・サイズ宣言部ではノード変数 (またはスイッチ変数) のインデックスに範囲付変数を用いることができ、これにより複数のノード変数 (スイッチ変数) を一括して宣言することができる。逆に、ノード変数 (スイッチ変数) の入力チャネルと出力チャネルを個々に宣言することもでき、これにより細かなスイッチ・サイズの宣言を行なうことができる。

バーチャル・チャネルを用いたい場合には、スイッチ・サイズ宣言部を以下のように記述する。

```
switch_size
    変数 {i(入力数),o(出力数)}#v (チャネル数), … ;
```

#v はチャネルのバーチャル化を表す予約語であり、それに続く括弧内の数値がバーチャル・チャネルの本数を表す。チャネルのバーチャル化は個々のノードもしくはスイッチに対して行うことができる。

3.3 結合記述部

結合記述部では、各ネットワーク資源における入力チャネルと出力チャネルの結合を記述する。結合記述部の構成は次のようになる。

```
connection{
    出力チャネル | 入力チャネル;
                  ⋮
}
```

結合演算子 | により左辺の出力チャネルと右辺の入力チャネルを結合する。入力チャネルと出力チャネルはそれぞれノード (またはスイッチ) の入力チャネルと出力チャネルを表し、ノード (またはスイッチ) 変数名と入力 (出力) チャネルを表す予約語 **i** (または **o**) とセットにして表される。また、ノード変数、スイッチ変数、入力チャネル、出力チャネルのインデックス内に範囲付変数を含む式を用いることができる。ここでは範囲

付変数は特別な働きをする。例えば、範囲付変数 i を用いて次のような結合が記述されるとする。

```
pu(i).o(0) | pu(i+1).i(1);
```

この時、右辺の範囲付変数は左辺の範囲付変数と対応し、どちらの範囲付変数も同じ値が適用される。例えば、左辺の i で定数 a という値が適用された場合、右辺の i もそれに対応し a という値をとり、結果として pu(a+1) が参照される。

範囲付変数のこの機能は、超並列計算機システムを簡略化した表現で記述する際に有効な機能である。

3.4 ルーティング・アルゴリズム記述部

ルーティング・アルゴリズム記述部では、各ノードおよびスイッチにおけるルーティング・アルゴリズムを記述する。その構成は次のようになる。

```
routing{
    ネットワーク資源{
        ルーティング・アルゴリズム
    }
                ⋮
}
```

この記述中のネットワーク資源とはノードもしくはスイッチを配列表現し、そのインデックスを範囲付変数により表したものである。ここでは、そのネットワーク資源におけるルーティングを C 言語の記述を用いて、自由に手続き的にプログラムすることができる。必要に応じ、このブロック中でローカル変数を宣言し、使用することもできる。また、ルーティング・アルゴリズム記述部ではルーティングの決定あるいは決定の保留を行なうことができ、それぞれ **route**、**retry** という命令により行なう。

- route(p,v): そのネットワーク資源における出力先を決定する命令である。p は物理チャネルの出力先、v はバーチャル・チャネルの出力先を表す。チャネルのバーチャル化を行なっていない時には、v を指定する必要がなく p のみを指定すればよい。

- retry: 現在のシミュレーション・クロックではルーティングを決定せず、保留する。この場合、その時刻における出力先の決定は中止され、次の時刻に再びルーティング・アルゴリズムが評価される。

一度 route 命令により決定された出力先は、メッセージが次のネットワーク資源に進まない限り変更されない。そのためネットワークの混雑度により出力先を変えるような動的なルーティングを行なう場合、必要に応じて **retry** 命令を用いる。

4 NDL の記述例

本節ではハイパキューブ (HC)、トーラス (TR)、ハイパクロスバ・ネットワーク (HXB) といったいくつかのネットワークに対する NDL の記述例を示す。NDL 内では関数 net_config() を定義する必要があるが、紙面の都合上この関数の記述は省く。

4.1 ハイパキューブ

HC の記述例を図 2 に示す。この記述例では任意の次元を表現するために、PU 空間を 1 次元で表している。この記述中の変数 n、dim はそれぞれ PU 数、次元数を表し、これらは net_config() によりシミュレーション実行時に与えられる。

1 つの PU は、アドレスの 0 から dim-1 番目のビットのうち 1 つのビットを反転させたアドレスを持つ PU と結合される。結合記述部では範囲付変数 i,j とビット演算子によりこれを行なっている。

ルーティング・アルゴリズムは e-cube ルーティングを用いる。現在そのメッセージの存在する PU のアドレスと転送先 PU のアドレスを下位ビットから比較していく。あるビットが異れば、そのビットの表す方向へ出力先を決定する。これを繰り返して、現在の PU と転送先 PU のアドレスが一致すれば転送は終了する。NDL では転送先 PU のアドレスを調べるために msg_dest_dim() というシステム関数を用意している。この関数は引数を 1 つ取り、転送先 PU のその引数が与える次元のアドレスを返す関数である。

```
var i in [0:n-1], j in [0:dim-1];
resource{
    node pu(n);
    switch_size
        pu(i){i(dim),o(dim)};
}

connection{
    pu(i).o(j) | pu(i^(1<<j)).i(j);
}

routing{
    pu(i){
        int k;

        for(k=0; k<dim; k++)
            if((i&(1<<k))
                != (msg_dest_dim(0)&(1<<k)))
                route(k);
    }
}
```

図 2: ハイパキューブの記述例

4.2 トーラス

単方向 2 次元 TR の記述例を図 3 に示す。PU は 2 次元の X × Y の構成になる。

単方向 2 次元 TR は各次元の隣接 PU の一方に対して結合されるので、次元数の 2 が入出力数を表す。デッドロック・フリーを保証するために 2 本のバーチャル・チャネルを用意する [5]。

出力チャネルのうち番号0のものはX方向、1のものはY方向の転送にそれぞれ使われる。PU間の結合はアドレスの大きい方から小さい方へ向けて結合される。また、modulo演算子'%'を用いてリング状の結合を記述している。

ルーティング・アルゴリズムはXYルーティングである。まず、メッセージの存在するPUのアドレスと転送先PUのアドレスをX次元方向で比較し、異っていればX次元方向への転送を行なう。X次元方向のアドレスが一致したら続いてY次元方向の転送に移る。route文では、チャネルのバーチャル化を行なっているため、物理チャネルとバーチャル・チャネルの両方を指定しなければならない。トーラスでwormhole方式の転送を行なうとデッドロックの可能性があるため、この記述例ではDallyのアルゴリズム[5]に基づいてデッドロック・フリーを保証している。現在のPUのアドレスと転送先PUのアドレスを比較し、転送先PUのアドレスの方が大きい場合はhiチャネルを用い、小さい場合はlowチャネルを用いる。

```
#define hi  1
#define low 0
var i in [0:X-1], j in [0:Y-1];
resource{
    node pu(X,Y);
    switch_size
        pu(i,j){i(2),o(2)}#v(2);
}

connection{
    pu((i+1)%X,j).o(0) | ex(i,j).i(0);
    pu(i,(j+1)%Y).o(1) | ex(i,j).i(1);
}

routing{
    pu(i,j){
        if(i != msg_dest_dim(0)){
            if(i < msg_dest_dim(0))
                route(0,hi);
            else
                route(0,low);
        }
        if(j != msg_dest_dim(1)){
            if(j < msg_dest_dim(1))
                route(1,hi);
            else
                route(1,low);
        }
    }
}
```

図3: 単方向2次元トーラスの記述例

4.3 ハイパクロスバ・ネットワーク

図4および5に3次元HXBの記述例を示す。この例ではPU配列は3次元のX × Y × Zの構成になる。

資源記述部ではPUはノード変数puとして宣言され、PU間を結合するスイッチとしてx_xb、y_xb、z_xb、exの宣言を行なっている。この記述例では各ネットワーク資源に対し、各チャネルを2重にバーチャル化している。HXBでは次元オーダのルーティングを行なう場合、デッドロックは起きない。ここでは、デッドロック・フリーのためにバーチャル・チャネルを用いるのではなく、転送性能の向上のために用いている。

資源記述部では、PUおよびスイッチの結合を簡略に記述するため、範囲付変数i, j,kを使用している。各PUの1次元方向はx_xbにより完全結合され、2次元および3次元はそれぞれy_xbおよびz_xbにより完全結合されている。これら3種のクロスバ・スイッチは直接PUとつながっているのではなく、exというルータを経てPUと結合される。ルータexも一種のクロスバ・スイッチである。

```
range Rx=[0:X-1], Ry=[0:Y-1], Rz=[0:Z-1];
var i in Rx, j in Ry, k in Rz;
resource{
    node pu(X,Y,Z);
    switch ex(X,Y,Z), x_xb(Y,Z),
         y_xb(Z,X), z_xb(X,Y);
    switch_size
        pu(i,j,k){i(1),o(1)}#v(2),
        ex(i,j,k){i(4),o(4)}#v(2),
        x_xb(j,k){i(X),o(X)}#v(2),
        y_xb(k,i){i(Y),o(Y)}#v(2),
        z_xb(i,j){i(Z),o(Z)}#v(2);
}

connection{
    pu(i,j,k).o(0) | ex(i,j,k).i(3);
    ex(i,j,k).o(0) | x_xb(j,k).i(i);
    ex(i,j,k).o(1) | y_xb(k,i).i(j);
    ex(i,j,k).o(2) | z_xb(i,j).i(k);
    ex(i,j,k).o(3) | pu(i,j,k).i(0);
    x_xb(j,k).o(i) | ex(i,j,k).i(0);
    y_xb(k,i).o(j) | ex(i,j,k).i(1);
    z_xb(i,j).o(k) | ex(i,j,k).i(2);
}
```

図4: 3次元ハイパクロスバの記述例
(資源記述部、結合記述部)

ルーティング・アルゴリズム記述部では物理チャネルに関して、1次元、2次元、3次元の順序の次元オーダのルーティングを行なう。puではまずメッセージをルータexへ出力する。x_xb、y_xb、z_xbではそれぞれその次元における転送先PUのアドレスを調べ、それに対応する出力先を選ぶ。exではまだ転送されていない次元に対応するクロスバ・スイッチへの転送を行なう。その際のクロスバ・スイッチの優先順位はx_xb、y_xb、z_xbの順であり、不必要な次元への転送は行なわない。

バーチャル・チャネルの出力先の決定に関しては、swt_phys_free()、swt_virt_free()という2つのシステム関数によりそれぞれ出力先の物理チャネル、バーチャル・チャネルの混雑度を調べ、2本用意されているバーチャル・チャネルのうち空いているチャネルを利用する。また、この記述中の関数msg_trail_move()はそのメッセージの先頭以降の部分が動けるのかどうかを調べるシステム関数であり、この記述ではもしメッセージが進めない場合はその時刻における出力決定は行なわない。このように種々の関数を用いて確実にそのメッセージが進める場合のみ出力先を決定しており、進めない場合はretry命令によりその時刻において、出力先の決定は行なわないようにしている。

NDLではこのように非常に複雑な動的ルーティング・アルゴリズムを記述することもできる。

```
routing
{
    pu(i,j,k){
        if(swt_phys_free(0)){
            if(swt_virt_free(0,0))
                route(0,0);
            if(swt_virt_free(0,1))
                route(0,1);
        }
        retry;
    }
    x_xb(j,k){
        int i;

        if(!msg_trail_move())
            retry;
        i=msg_dest_dim(0);
        if(swt_phys_free(i)){
            if(swt_virt_free(i,0))
                route(i,0);
            if(swt_virt_free(i,1))
                route(i,1);
        }
        retry;
    }
    y_xb(k,i){ "x_xb と同様" }
    z_xb(i,j){ "x_xb と同様" }
    ex(i,j,k){
        if(!msg_trail_move())
            retry;
        if(msg_dest_dim(0) != i){
            if(swt_phys_free(0)){
                if(swt_virt_free(0,0))
                    route(0,0);
                if(swt_virt_free(0,1))
                    route(0,1);
            }
            retry;
        }
        if(msg_dest_dim(1) != j){
            if(swt_phys_free(1)){
                if(swt_virt_free(1,0))
                    route(1,0);
                if(swt_virt_free(1,1))
                    route(1,1);
            }
            retry;
        }

                    ⋮

        retry;
    }
}
```

図 5: 3 次元ハイパクロスバの記述例
(ルーティング・アルゴリズム記述部)

5 シミュレーションによる性能評価

本節では、INSPIRE を用いてネットワークの転送性能の評価を行なう。直接網としてハイパキューブ (HC)、双方向トーラス (TR)、間接網としてハイパクロスバ・ネットワーク (HXB) を取りあげる。PU 数は 1024 とし、HC は 10 次元、TR は 2 次元と 3 次元でそれぞれ 32× 32 と 8 × 8 × 16 の構成、HXB は 3 次元で 8 × 8 × 16 の構成とする。ただし、これらネットワークの各 PU にルータを置くものとする。従って、HC および TR は直接網であるが NDL の記述中では間接網のようにスイッチを表すスイッチ変数を宣言し用いている。

シミュレーションに与える条件としてはリンクのバンド幅 1 [byte/clock] 、総シミュレーション時間 2000 [clock]、メッセージ長は固定で 10 [byte] とする。メッセージ生成確率をパラメータとして変化させ、ネットワークの負荷を変えて評価を行なう。対象とする転送パターンはランダム転送であり、PBF 内で記述されている。HC、TR、HXB に対しそれぞれ同じ PBF を使用しており、同じ転送パターンによる性能評価を行なっている。

5.1 ネットワーク・トポロジ間の比較

HC、TR、HXB の転送性能の比較を行なう。HC および HXB はチャネルのバーチャル化を行なわない。しかし、TR はネットワークそのものに潜在的なデッドロックの可能性を秘めているため、チャネルをバーチャル化しデッドロック・フリーを保証している [5]。

図 6 にシミュレーション結果を示す。縦軸は平均レイテンシで横軸はスループットである。レイテンシはメッセージが PU 内で生成されてから、転送先に到達するまでの時間であり、スループットは各 PU のメッセージの総受信量を、各 PU が 1byte/clock でメッセージを受信した場合を 1 として正規化している。

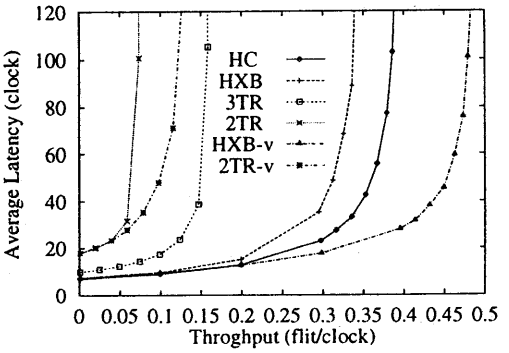

図 6: ランダム転送におけるシミュレーション結果

図中のレイテンシが急激に上昇する点がネットワークの飽和点であり、その時点でのスループットが最大スループットである。スループットがほぼ0 の時、つまりネットワークにかかる負荷がほとんどない時、平均レイテンシは平均距離を示す。図 6 より各ネットワークの平均距離は HC が 7、2 次元 TR が 18、3 次元 TR が 10、HXB が 7.38 となっている。

TR は次元が大きくなれば平均距離が縮まり、それに伴い転送性能も向上する。また、TR は次元を大きくすればネットワーク自体が HC に近付く。従って、2 次元 TR から 3 次元 TR、HC へと転送性能が向上している。HXB は次元数はそれほど大きくないものの、各次元方向の PU はクロスバ・スイッチにより完全結合されており、各次元での転送は 1 ステップで済む。そのため、HC 程ではないが高い転送性能を実現できている。

5.2 チャネルのバーチャル化による性能向上

2次元TR、3次元HXBに対し、それぞれ性能を向上させるためにチャネルをバーチャル化し、性能向上の度合を調べる。図6にシミュレーション結果を示す。

TRではAP1000[1]のようにステップ数毎に使用するチャネルを変えるというルーティング・アルゴリズムを用いる(図6の2TR-v)、そのために各チャネル毎に各次元方向の半分のサイズの多重化が必要となる(双方向のため)。HXBでは図4および5で示したように、チャネルを2本に多重化し、ネットワークの混雑度により使用するチャネルを変えるというルーティング・アルゴリズムを用いる(図6のHXB-v)。

チャネルのバーチャル化によって、それぞれ2TRは113%、HXBは40%の転送性能の向上が見られる。2TRは2倍以上の転送性能を実現できているが、8倍のバッファ容量を必要とする。もともと2TRは単方向リンク当り2本のバーチャル・チャネルを用いており、ステップ毎に使用するバーチャル・チャネルを変えるという方法では、16本のバーチャル・チャネルを必要とする。そのため8倍のバッファ容量を用意する必要がある。一方、HXBはバッファ容量を2倍とするだけで1.4倍の転送性能を実現でき、しかもHCを上回る程の性能向上が見られる。

5.3 シミュレータの実行速度

INSPIREにより生成されるシミュレータと専用シミュレータの実行速度の比較を行なった。その結果、INSPIREにより生成されたシミュレータに約40%の速度低下が見られたが、この程度の速度低下はINSPIREの汎用性を考えると実用範囲であると考えられる。

6 おわりに

本稿では、汎用ネットワーク・シミュレータ生成系INSPIREを提案し、実際にトランスレータ、カーネルなどの作成を行ない、これを用いた各種ネットワークの特性の比較について報告した。このシステムでは、ユーザがネットワーク記述専用言語NDLによりネットワークの資源、ネットワーク・トポロジ、ルーティング・アルゴリズムといったネットワークの諸特性を記述することができる。NDLでは範囲付変数という概念を導入しており、ネットワークの記述の簡略化を行なうことにより、超並列ネットワークを容易に記述できる。本稿では、NDLにより様々な直接網・間接網が記述できることを確認した。さらに、NDLでは出力先のチャネルの混雑度を調べ、それにより出力先を変えるという動的なルーティング・アルゴリズムも記述できることを確認した。

また、PUの動作はC言語により手続き的に記述することができ、これにより簡単な転送パターンから複雑な転送パターンまで自由に記述することができる。PUの動作を記述したファイルであるPBFと、ネットワークをNDLにより記述したファイルであるNDFはそれぞれ独立に記述することができる。従って、同じネットワークにおいて様々な転送パターンによる性能評価を行なうことも、逆に同じ転送パターンにおける、様々なネットワークの性能評価を行なうこともできる。

INSPIREの今後の課題としては、NDLを拡張することが考えられる。現在、メッセージ転送方式としてwormhole方式のみをサポートしている。これに対し、より複雑な転送方式であるvirtual cut-through[6]をサポートし、様々なバッファサイズのネットワークを記述できるようにする予定である。さらに今後の課題として、ネットワーク中の各資源の利用率等の統計情報の自動収集がある。これにより各チャネルの負荷の偏りを調べることができ、さらに細かな性能評価を行なうことができる。また、NDLはネットワークを比較的容易に記述することができるが、複雑なネットワークを記述する際には、やはりユーザの記述ミスが多くなってしまう。現在、INSPIREではこのユーザのミスを検出する機能に乏しく今後これを改善する必要がある。

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中澤研究室並列処理研究グループ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(07NP0401)及び文部省科学研究費(奨励(A)07780222)の補助によるものである。

参考文献

[1] H. Ishihata, et. al., "An Architecture of Highly Parallel Computer AP1000", IEEE Pacific Rim Conference on Communication, Computers and Signal Processing, 1991

[2] 西川順次, "並列計算機ADENARTにおける各種相互結合網の実現", 並列シンポジウムJSPP'92, 1992

[3] 中澤喜三郎 他, "CP-PACSのアーキテクチャの概要", 情報研報ARC-108-9, October 1994

[4] 柴村英智 他, "超並列計算機のための相互結合網シミュレータ", 情報処理学会論文誌, April 1992

[5] W. J. Dally et. al., "Deaklock-Free Message Routing in Multiprocessor Interconnection Networks", IEEE Transaction on Computers, Vol.C-36, No.5, May 1987

[6] Parviz Kermain et. al., "Virtual Cut-Through: A New Computer Communication Switching Technique" Computer Networks vol3, No 4, 1979