6　並列計算技術の応用
6 Applied Parallel Processing Technology

6-1　分散処理言語 Overlay GHC とその応用可能性
6-1 Distributed Processing Language Overlay GHC and its Application Possibilities

斉藤賢爾
SAITO Kenji

要旨

近年のコンピュータの性能向上及びネットワークの高速化がオーバレイネットワークの高度な応用を可能にしていることに加え、マルチコア・プロセッサの利用が普及していることから、ソフトウェアシステムにおいて我々が取り扱うべき並行性の水準が急激に上昇している。超並行性の記述を自然に行え、密結合及び疎結合マルチプロセッサ環境の両方で並列実行できるプログラミング言語の登場が望まれている。

この稿では、そのような言語の候補として、いまだ開発途上にある Overlay GHC 言語と、特にトレーサブルネットワークにおけるその応用可能性を説明する。この言語は並行論理型言語 GHC[1]（Guarded Horn Clauses：ガード付きホーン節）に基づいている。

Today's high-performance computers and high-speed networks allow sophisticated applications of overlay networks. Meanwhile, usage of multi-core processors has been spreading. The level of concurrency we need to handle in software systems has been rising rapidly, which necessitates a language that can express massive concurrency in a natural way, which can work with both tightly and loosely-coupled multiprocessor environments in parallel executions.

This article describes still work-in-progress design of Overlay GHC, an overlay network programming language based on concurrent logic language GHC[1] (Guarded Horn Clauses), as a candidate for such a language, and its application possibilities especially in traceable networks.

[キーワード]
プログラミング言語，並行プログラミング，論理プログラミング，オーバレイネットワーク，並列推論
Programming language, Concurrent programming, Logic programming, Overlay network, Parallel inference

1　まえがき

現在、コンピュータシステムにおける並列実行環境、すなわち、複数プロセッサによるプログラムの同時実行のための環境が、高度化かつ広範囲で利用可能となってきている。このことに伴い、ソフトウェアシステムにおいて我々が取り扱うべき並行性、すなわち、複数プロセスによる論理的並列性の水準に変化が起きている。

その一つの例として、個々のコンピュータが、

大容量記憶装置や高速なプロセッサに代表されるようにふんだんな余剰資源を持ち、かつ高速ネットワークで結ばれることにより、P2P（peer-to-peer）やグリッドコンピューティングに代表されるような、オーバレイネットワークを用いたインターネットの高度な応用が現実のものとなっている。

P2P システムの設計の特徴は、参加者が互いに対称的であり、役割が動的に交換可能であるようなオーバレイネットワークの利用が行われている点にある。そのようなシステムの設計においては、自律エージェントの集合としてシステムを記述する必要があることから、高い水準の並行性を扱うことが要求される。また、グリッドコンピューティングでは、P2P のような自律性は重んじられないものの、分散的に配置されたコンピュータにおける余剰資源の発見と活用を動的に行う必要があり、やはり設計においては高い水準の並行性の取扱いが要求される。

一方、すべての主要 CPU 製造者は、シングルコアのプロセッサでの性能向上がもはや見込めないため、マルチコア・アーキテクチャへの移行を図っている。マルチコア・プロセッサの利用は既に広く浸透を始めているが、その性能を余すことなく引き出すためには、ソフトウェアにおいて高い水準の並行性を取り扱うことが要求される。

このような状況の下、ソフトウェアシステムにおいて我々が取り扱うべき並行性の水準は急激に上昇しており、超並行性を自然に記述できるプログラミング言語の出現が期待されている。そのような言語で記述された並行プログラムの並列実行は、マルチコア・プロセッサに代表されるような密結合マルチプロセッサ環境及び P2P やグリッドコンピューティングのような疎結合マルチプロセッサ環境が併存することはもはや前提であるため、その両方に対応する必要がある。

この稿は、そのような超並行性・並列性記述言語として、いまだ開発途上にある Overlay GHC 言語の設計と、特にトレーサブルネットワークにおけるその応用可能性について述べる。この言語は、個々のノードがマルチコア・プロセッサであることを前提としたオーバレイネットワークのプログラミングのための言語であり、並行論理プログラミング言語 GHC[1]（Guarded Horn Clauses：ガード付きホーン節）に基づいている。Overlay GHC はオーバレイネットワークを鳥瞰的視点から素早くプログラミングすることを可能にし、論理ネットワーク全体を一つのプログラムとして記述できることを特徴とする。

2 背景

ここでは、GHC の構文及び意味、そしてGHC の密結合マルチプロセッサ環境向け拡張である KL1 言語[2] について説明する。

2.1 GHC：Guarded Horn Clauses（ガード付きホーン節）

GHC は上田和紀博士により設計され、第五世代コンピュータプロジェクトを推進した ICOT（新世代コンピュータ技術開発機構）の研究の一環として1985 年に発表された並行論理型言語である。

2.1.1 前準備

GHC プログラムの基本的な単位は「項（term）」である。項は、関数記号と変数から構成される。関数記号は英小文字から始まる記号であり、変数は英大文字又は“_”から始まる。変数は単独で項である。それ以外の場合、項は次の形式をとる。

$$f(\gamma_1, \gamma_2, ..., \gamma_n) \ (n \geqq 0)$$

ここで f は関数記号であり、一連の γ は項である。n = 0 のとき、f は特に定数と呼ばれる。例えば、“X”，“a”，“cons（a, X）”，“cons（a, cons（b, nil））”はすべて項である。「アトム（atom）」（原子論理式；atomic formula）は次の形式をとる。

$$p(\gamma_1, \gamma_2, ..., \gamma_n) \ (n \geqq 0)$$

ここで p は述語記号であり、一連の γ は項である。述語記号は英小文字から始まる記号である。例えば、“is_list（cons（a, X））”はアトムである。

2.1.2 GHCの構文

GHC プログラムは、次の形式をとるガード付きホーン節の集合である。

$$H \leftarrow G_1, ..., G_m \mid B_1, ..., B_n. \ (m, n \geqq 0)$$

ここでH及び一連のGとBはアトムであり、“|”は「コミット(commit)演算子」と呼ばれる。Hは特に「頭部(head)」と呼ばれる。一連のGとBは「ゴール(goal)」と呼ばれ、それぞれ「ガードゴール(guard goal)」「ボディゴール(body goal)」と区別されることがある。ゴールは、次の形をとる「同一化ゴール(unification goal)」である場合がある。

$$\gamma_1 = \gamma_2$$

ここで γ_1 及び γ_2 は項である。Hと一連のGを合わせたものを「ガード(guard)」と呼び、一連のBを「ボディ(body)」と呼ぶことがある。ガードやボディにゴールが無い場合、“true”を置く。プログラムは次の形式をとるゴール節により起動される。

$$\leftarrow B_1, \ldots, B_n. \quad (n \geqq 0)$$

ここで一連のBはゴールである。

2.1.3 GHCプログラムの宣言的意味

ガード付きホーン節は次のように読むことができる。

ガードとボディのすべてのゴールが真であれば、頭部は真である。

成功したGHCプログラムの実行結果は、必ずこの意味にのっとる(健全性)。ただし、実行において一部の節が適用されないこともあるため、その結果は唯一の解でない場合がある(不完全性)。

2.1.4 GHCプログラムの操作的意味

直観的には、各ガード付きホーン節はゴールの書換え規則を表す。ガードが規則の適用のために満たすべき条件を指定し、ボディが実際に書換えに用いるゴール群を指定すると見なせる。もし、複数の節が適用可能であれば、一つの節のみが非決定的に選択される。節の適用は非可逆的な操作であり、「コミットメント(commitment)」と呼ばれる。変数への代入は「束縛(binding)」と呼ばれる。束縛は、コミットメント後に行われ、束縛済みの変数に対して非互換な項を束縛しようとする試みは必ず失敗する。変数でない項への束縛は特に「具体化(instantiation)」と呼ばれる。二つの項は、その中に出現する変数に、互いの対応する項を束縛することで字句的に同一になったとき、「同一化された(unified)」と言う。

GHCプログラムの操作的意味を以下に簡易に示す。

1 ゴールの実行：ゴール節の各ゴールは以下のステップで並行に実行される。

(a) 頭部同一化(head unification)：節の頭部に現れる変数は、手続き型言語の仮引数に相当する。呼び出し元のゴールと同一化が可能な頭部を持つすべての節が同時にコミットメントの候補となり、その頭部に現れる変数は、呼び出し元のゴールの対応する項に束縛される。

(b) 中断規則(suspension rule)：呼び出し元のゴールに現れる変数を具体化するようなすべての試みは中断される、という制限の下で、候補節のガードゴールは並行に実行される。

(c) コミットメント(commitment)：実行は、ガードが成功した候補節の一つにコミットし、呼び出し元のゴールは、コミットされた節のボディにより置き換えられる。当該ボディの同一化ゴールは、呼び出し元のゴールに現れていた変数を具体化できる。

2 成功：同一化ゴールは、その引数が同一化できるとき成功する。その他のゴールは、最終的に、成功する同一化ゴールで置き換えられるか、あるいは空のゴールで置き換えられるとき、成功する。プログラムは、ゴール節内のすべてのゴールが成功したとき、成功する。

3 失敗：同一化ゴールは、その引数が同一化できないことが確実になったとき失敗する。その他のゴールは、コミットできる節の候補が無いか、あるいはすべての候補のガードが失敗したとき、失敗する。プログラムは、ゴール節内のゴールがどれか一つでも失敗したとき、失敗する。

2.1.5 GHCプログラムのプロセス的解釈

GHCプログラムは、次のようにプロセスの集合である並行プログラムとして解釈できる。

1 再帰的に定義される述語がプロセスを定義する。

2 プロセスの連結がプロセス間ネットワークを定義する。

3 ゴールの引数がプロセスの内部状態を定義する。
4 ゴール間で共通に現れる変数が通信チャンネルを定義する。これらの変数は、多くの場合ストリーム（データ列）を表現するために用いられる。

図1は、[3]での例に基づいてGHCで書かれたスタックのプログラムである。

“：−”は“←”のコーディング上の表現であり、“[a | b]”，“[a，b]”及び“[]”は、それぞれ“cons(a，b)”，“cons(a，cons(b，nil))”及び“nil”の構文シュガーである。“io：printstream(Os)”は筆者らのOverlay GHC実装で提供される組み込みプロセスであり、ストリーム“Os”の要素を表示する。このプログラムは“[2,1]”，“2”，“[1]”，“1”，そして“[]”という列を生成する。

論理プログラミングでの慣習に倣い、述語を「述語名/引数の数」と表記すると、この例では、ゴール節において、完成されたリストをコマンド列としてstack/3プロセスに渡している。しかし、consのCDR部（リストの残り）が変数であるような未完成のリストを用いることで、新たなコマンドが到着するのを待って処理を行うようなデータ駆動プロセスとしてstack/3を利用できる。

2.2 KL1：Kernel Language One（核言語1）

KL1はGHCの拡張であり、第五世代コンピュータプロジェクトにおける並列推論マシンのシステム記述言語として、近山隆博士らにより設計された。GHCプログラムが並行性のみを記述するのに対し、KL1プログラムでは並列性も記述できる。すなわち、プロセスのプロセッサへの割当て等、並行プログラムをいかに並列実行するか、ということについても記述できる。ただし、これによりGHCプログラムの意味が変化することはなく、KL1プログラムからプラグマ（言語処理系への指示）を取り除くことにより、対応するGHCプログラムが得られるという、並行性と並列性の明示的な分離が行われている。

3 Overlay GHCの設計

3.1 方針

筆者は過去に、分散リアルタイム環境向けにGHCの拡張を行ったことがある[4]。このときのアプローチは、ゴールの投入からコミットメントの発生までの遅延に対し、時間制約を記述する新たな「時間付きガード部」を導入するというもので、GHCプログラムの意味を、一部、変更する必要があった。

Overlay GHCの設計では、これと異なり、KL1と同様のアプローチをとることにした。すなわち、GHCプログラムを分散リアルタイム環境で実行する方法は、並列性への言及であるとして基本的にプラグマにより記述し、GHCの言語仕様自体には手を加えないことにした。これにより、並行性と並列性の明示的な分離による、処理系依存部の抽出やプログラムの正当性の判断の容易さといった効果に加え、GHCプログラムに関するプログラム変換等の過去の研究成果や、GHCで書かれたプログラムの蓄積そのものなどが、直接的に利用可能になることが期待できる。

3.2 プラグマ

現状、Overlay GHCでは表1に示すプラグマを定義している。

このうち、ゴールの実行順序や節の適用の優先順位に関するプラグマについては、KL1と共通になっている。Overlay GHCを特徴付けるのは、

```
% 述語 stack(コマンドストリーム,データ,出力ストリーム)を定義。
stack([pop|Cs], [X|List], Os) :-true | Os = [X|Os1], stack(Cs, List, Os1).
stack([push(X)|Cs], List, Os) :-true | stack(Cs, [X|List], Os).
stack([list|Cs], List, Os) :-true | Os = [List|Os1], stack(Cs, List, Os1).
stack([], List, Os) :-true | Os= [].

:- stack([push(1), push(2), list, pop, list, pop, list], [], Os), io:printstream(Os).
```

図1 例：GHCで書かれたスタックのプログラム

表1　Overlay GHC におけるプラグマ一覧

種類	名称	意味	備考
実行順序	@priority(Level)	指定された優先順位でゴールを実行する。	KL1 と共通
	@lower_priority	低い優先順位でゴールを実行する。	
	alternatively.	以降の節を適用する優先順位を低くする。	
ゴール配置	@this_node	ゴール節を投入したノードでゴールを実行する。	Overlay GHC 特有
	@other_node	ゴール節を投入したノード以外でゴールを実行する。	
	@node_id(ID)	ID で指定されたノードでゴールを実行する。	
実時間制御	@periodic(Msec)	指定された間隔（ミリ秒）でゴールを実行する。	

ゴール配置と実時間制御に関するプラグマである。

KL1 では、“@node（ノード番号）”プラグマによりゴールをプロセッサに配置することが可能である。これは、プロセッサの個数が固定的である密結合マルチプロセッサ環境向けの仕様と言える。対して、Overlay GHC では、複数プロセッサの構成が動的に変化する環境を想定し、ゴール節の投入が行われたノードに対する相対でノードを指定するプラグマ及び識別子による遠隔ゴール配置プラグマ“@node_id（ID）”を導入した。ここで、“ID”は URL 形式で記述される識別子である。例えば、“xmpp ://”スキームにより Jabber ID を指定したり、“pgp ://”スキームにより PGP 公開鍵ユーザ ID を指定したりすることを想定し、言語処理系はこれらの識別子を実際の通信の対象にマップする機構を持つものとしている。

また、GHC/KL1 では、コンソールが 1 台であるような、単体の並列推論マシンを前提にしていたため、io：outstream/1 などのコンソール入出力ストリームにかかわる組み込みプロセスを、一つのプログラムにつき、たかだか 1 回しか用いられないとしていた。

一方、Overlay GHC では、各ノードが独立したコンピュータである疎結合マルチプロセッサ環境を想定している（ただし、各コンピュータは多くの場合マルチコア・プロセッサを持つ密結合マルチプロセッサ環境である）ため、コンソール入出力ストリームに係わる組み込みプロセスはノードごとに生成可能としている。これにより、図 2 に示すような、各参加者が最寄りの端末で入出力を行うチャット型のオーバレイネットワーク等を一つのプログラムとして記述することが可能になる。また、P2P システム等の開発時のシミュレーションにおいて、1 台のコンピュータの上で複数の仮想ノードを生成し、ノードごとにウィンドウを区別して端末からの入出力を行うことも可能になる。

周期実行プラグマ“@periodic（ミリ秒）”は、ゴールが投入されてからコミットメントが行われるまでの時間間隔を指定するものである。再帰プロセスとして記述されているゴールは、これにより、指定された間隔で周期実行される。この機能は、例えばセンサの値をポーリングするような場合に用いられる。

3.3　遠隔ゴール配置（RGP：Remote Goal Placement）

Overlay GHC でのゴール配置プラグマは、プログラムのマイグレーションを自動的に行う。すなわち、配置先のノードが該当するプログラムを持っていない場合、ゴールの配置に先駆けて、プログラムが相手側ノードに自動的に転送される（これに関するセキュリティ上の議論については**3.6** で述べる）。このことを用いた「遠隔ゴール配置（RGP：Remote Goal Placement）」は、OverlayGHC におけるプログラミングを特徴付けるテクニックの一つである。このテクニック

```
terminal(NickName, InStream, OutStream) :-true
  |  keyboard(NickName, InStream, OutStream),
     display(NickName, OutStream, DisplayStream),
     io:outstream([write('terminal started.'), nl|DisplayStream]).

keyboard(NickName, InStream, OutStream) :-io:read(X)
  | checkInput(NickName, InStream, OutStream, X).
keyboard(NickName, [Term|InStream], OutStream) :-Term \= line(NickName, _)
  | OutStream = [Term|Xs], keyboard(NickName, InStream, Xs).
keyboard(NickName, [line(NickName, X)|InStream], OutStream) :-X \= left
  | keyboard(NickName, InStream, OutStream).
keyboard(NickName, [line(NickName, left)|InStream], OutStream) :-true
  | InStream = OutStream.

checkInput(NickName, InStream, OutStream, X) :-Line := NickName + ' : ' + X
  | OutStream = [line(NickName, Line)|Xs], keyboard(NickName, InStream, Xs).
checkInput(NickName, InStream, OutStream, join(Name, ID)) :-true
  | terminal(Name, InStream, Xs)@node_id(ID), keyboard(NickName, Xs, OutStream).
checkInput(NickName, InStream, OutStream, leave) :-true
  | OutStream = [line(NickName, left)|Xs], keyboard(NickName, InStream, Xs).
checkInput(NickName, InStream, OutStream, X) :-otherwise
  | keyboard(NickName, InStream, OutStream).

display(NickName, [line(_, Line)|OutStream], DisplayStream) :-Line \= left
  | DisplayStream = [write(Line), nl|Xs], display(NickName, OutStream, Xs).
display(NickName, [line(NickName, left)|OutStream], DisplayStream) :-true
  | DisplayStream = [write('I have left.'), nl].
display(NickName, [line(Someone, left)|OutStream], DisplayStream)
 :- NickName \= Someone, Line := Someone + ' has left.'
  | DisplayStream = [write(Line), nl|Xs], display(NickName, OutStream, Xs).

% Starts a group chat as the initiator whose nickname is 'foo'.
:-terminal('foo', X, X).
```

図2 Overlay GHC で記述した簡単なグループチャットのプログラム

により、分散プログラミング上の多くの興味深い概念を実現できる。筆者らは、遠隔手続き呼び出し(RPC)との意味的な相違から、これを利用した、これまでと異なるネットワークプログラミングの文化が開花することを期待している。

図2は、このテクニックを用いて記述した、対称的(サーバを持たない)グループチャットのプログラムである。

新しいユーザは、招待ベースでチャットに参加できる。プログラム中、“\=”は同一化不可能を示す組み込み述語であり、“otherwise”は、他のすべての候補節のガードが失敗したときに成功する組み込み述語である。このプログラムでは、terminal/3 がユーザを表現する主プロセスである。keyboard/3 は、他のユーザからのものも含めて入力を受け付けるプロセスであり、受け付けた入力を、環状オーバレイネットワーク上の次のユーザに転送する。checkInput/4 は keyboard/3 に従属し、コマンドを解釈する。ユーザからの入力が“join(Name, ID)”という形式だった場合、RGP を用い、新しい terminal/3 プロセスを遠隔ノード上に生成させる。display/3 プロセスはコンソール出力ストリームを生成する。

3.4 障害の検出と取扱い

分散計算においては、プログラムが論理的に健全であっても、プロセッサや通信の障害により実行が失敗する場合がある。しかし、図2のプログラムは、障害の発生を前提にしていないため、修正が必要である。

筆者らは、障害の検出や取扱いを Overlay GHC で自然に記述するための方法を検討してきた。検討にあたっては、障害の発生によっても GHC プログラムの意味が変化しないことが重要である。RGP では、障害モデルをフェイルストップ[5]に限定することにより、ローカルなゴール配置と

RGP を、矛盾なく等価なものとして扱うことができる。

フェイルストップ障害モデルでは、障害が発生したプロセスは、いかなる副作用ももたらさずに計算を停止する。Overlay GHC では、このことは永続的に実行が中断されることを意味するが、ホーン節には時間の概念がないため、これにより GHC プログラムの意味が変化することはない（非常に実行が遅いプロセスとして扱うことができる）。

多くの障害は、広めに検出し、偽陽性（false positive）の場合、誤って障害が検出されたプロセスをわざと停止させることにより、フェイルストップとして扱うことができる。

筆者らは、ガードで利用できる組み込み述語として timeout/1 を導入し、プログラムがフェイルストップにより発生した永続的中断の状態を検出し、この状態から回復できるようにすることを検討している。

図 3 は timeout/1 を用いたプログラムの例である。このプログラムは、30 秒ごとに“ping(Pong)”を送出し、変数 “Pong”が 10 秒以上、具体化されなければ障害として検出する。

フェイルストップに落とし込めない障害モデルにビザンチン障害[6]がある。ビザンチン障害モデルでは、障害が発生したプロセッサは任意の項で変数を束縛し得る。この種の障害の検出のためには、冗長性をプログラムする必要がある。Overlay GHC では、ゴール節において、等価な複数のゴールを記述し、RGP を用いて別々の遠隔ノードに配置することで冗長性を簡易にプログラムできる。障害が発生し、等価な複数のゴール実行に関与した等しい変数に対し、矛盾する内容が束縛された場合、同一化が失敗し、ゴール節は AND 並列で実行されるため、そのプログラム全体が失敗する。このことをプログラム的に検出し、そこから回復するためには、述語のメタコール等の仕組みが必要となる。

3.5 動的な多対 1 通信

Overlay GHC では、変数の具体化は 1 回限りのものである。このことは、1 本のストリームに対して、複数のライタが同時に書き込むことを不可能にする。

並行論理型言語では、この種の困難を、n 本のストリームを入力し、マージした 1 本のストリームにする merge/（n＋1）プロセスを利用することで回避して来た。このプロセスは並行論理型言語自身により簡単に実装できる。しかし、オーバレイネットワークのプログラミングでは、より動的な多対 1 通信が支援される必要が生まれる。例えば、P2P システムにおいて、各ノードが動的に変化する近隣ノード群からメッセージを受け取るような場合である。この問題を解決するために、筆者らは「キュー（queue）」データ型の導入を検討している。図 4 に例を示す。

プログラム中、“<<”はキューへの追加を行う組み込み述語である。キューは、（主としてローカルノードで実行される）プロセスの入力ストリームを表現する変数を引数として生成される。

```
checkAlive(Stop, Os) :-timeout(30000)
  | Os = [ping(Pong)|Os1], checkPong(Pong, Stop), checkAlive(Stop, Os1).
checkAlive(stop, Os) :-true | process-failure-of-the-peer.

checkPong(pong, Stop) :-true | true.
checkPong(Pong, Stop) :-timeout(10000) | Stop = stop.
```

図3 Overlay GHC で書かれた ping プログラム

```
createStack(Q, Os) :-true | Q := createQueue(Cs), stack(Cs, [], Os).

stacker(Q, ...) :-true | Q << push(X), ..., stacker(Q, ...).

:-createStack(Q, Os), ..., stacker(Q, ...)@node id(ID1), stacker(Q, ...)@node id(ID2).
```

図4 単一ストリームへの複数ライタ

複数のライタが並行にキューにデータを追加することができ、その結果は、当該変数を最初の cons とする 1 本のストリームにマージされて出力される。このデータ構造は、ローカルノードにおいて実働する。ローカルノードにおいては書き込むストリームを表す変数を保持し、遠隔ノードからのデータが到着すると、セマフォ等を用いた排他制御により逐次的に当該ストリームにデータを書き込む。遠隔ノードにおいては、実働しているデータ構造への参照を稼働先のノード ID とキューの識別子により保持し、キューへの追加が起きると当該ノードへのデータ転送を行う。

3.6 セキュリティの考察

適切なセキュリティの設計が行われない場合、RGP はマルウェアの転送と実行に用いられる恐れがある。したがって、当然、RGP は、指定された遠隔ノードが、転送されつつあるゴールの実行を許諾する場合のみ許される必要がある。筆者らは、このことを実現する最善の方式を検討中である(筆者らによる現在の実装では、簡易に、ゴールを受け付ける相手の Jabber ID のリストを持つようにしている。)。

4 応用可能性

4.1 分散コンピュータ

オーバレイネットワークを動的にプログラミングできるOverlay GHC には様々な応用可能性が考えられるが、それらは「分散コンピュータ(distributed computer)」の概念により統一的に扱うことができる。

分散コンピュータとは、負荷の分散や、特定用途に向けた入出力のルーティングといった目的のために、計算機の部品を遠隔のプロセッサに配置することである。Overlay GHC では、RGP により、そのような機能の分散を簡潔に記述できる。

4.2 トレーサブルネットワークへの応用可能性

トレーサブルネットワークに向けたツールキットでの応用としては、GHC が元々、並列推論マシンのシステム記述言語を意図して設計された特徴を生かして、並列推論エンジンを記述することがあげられる。実際に、[7] で紹介されているように、GHC によるプロダクションシステム記述等の研究が過去に盛んに行われていたため、その成果を直接的に応用できると期待している。この推論エンジンは、過去の観測から得られた演繹の材料を元に、動的にルールを生成しつつ、インシデントの発生に対して、適切なアクションを推論する。このエンジンは、推論と調査の反復を実現できる必要がある。

分散処理言語としての特質の活用としては、複数の仮説を同時に検証できる推論エンジンの可能性を追求できるだろう。具体的な推論エンジンの設計に関しては、プロダクションシステムのみならず、様々な知識情報処理に関する、並行論理型言語の過去の研究における成果を積極的に活用しつつ進めたい。

また、Overlay GHC のトレーサブルネットワークへの応用では、このほかに、既存プロセスを組み合わせることによる糊言語(glue language)としての特徴を生かして、ツールキットの各部品をプロセスとして並列に接合することにも用いることができると考えている。

5 言語処理系の実装

Overlay GHC の言語処理系の例として、筆者らはオープンソースコミュニティにて Java 言語によるインタプリタを開発している。このインタプリタは、同様に筆者らが Java 言語で開発を進めている XMPP ベースのインスタントメッセージング・システムである wija のプラグインであり、最新版は以下の URL より取得できる。

- リリース版最新:http://www.media-art-online.org/ghc/
- 開発版最新:http://www2.media-art-online.org/nightly/

この Overlay GHC の実装には「パッケージプログラミングインタフェース(PPI:package programming interface)」が含まれる。これは、wija の他のプラグインが Overlay GHC プリミティヴやプロセスを提供するためのインタフェースであり、プログラマが既存の機能を Overlay GHC のプロセスとして連結・組み合わせることを支援する。

6　関連研究

6.1　分散KL1

分散 KL1 は、GHC を分散プログラミング向けに拡張するという意味で Overlay GHC の先行研究である。これは KLIC（KL1 から C 言語へのコンパイラ）の拡張であるDKLIC[8] 言語処理系として実装されている。

DKLIC はRGP に似た概念を持つが、それは「遠隔述語呼び出し」という用語で呼ばれ、むしろ RPC に近く、述語のマイグレーションは実装されていないように見える。単一ストリームへの複数ライタや、プロセッサ障害の問題には言及されていないようである。

6.2　P2/OverLog

P2[9] は OverLog 言語を用いてオーバレイネットワークを簡潔にかつ再利用可能な形式で記述するシステムである。

OverLog と Overlay GHC は同様な目標（宣言的で素早いオーバレイネットワークの記述）と同様なアプローチ（論理プログラミング）を共有している。両者が異なるのは、OverLog が Prolog のサブセットである DataLog に基づく問い合わせ言語であり、Overlay GHC は並行論理プログラミング言語の末裔であるという点である。これら二つの環境については、今後、更なる比較研究を予定している。

6.3　Erlang

Erlang[10] は超並行性の記述のために設計された関数型プログラミング言語である。

Erlang と Overlay GHC は、Erlang における並行計算がアクターモデル[11] に基づいており、このモデルは並行論理プログラミングの特殊な解釈の例である[12] という点において関係している。これら二つの言語についても、更なる比較研究を予定している。

6.4　StreamIt

StreamIt[13] は、高速なストリーミングアプリケーションのための言語である。この言語の設計には二つの目標がある。一つは、高水準なストリーミング抽象を提供することであり、もう一つは、ちょうど C 言語がフォン＝ノイマン型アーキテクチャにおける共通の高級アセンブリ言語であると見なせるように、グリッドベースのアーキテクチャにとっての共通の機械語／高級アセンブリ言語となることである。

ストリームプログラミングは、並行論理プログラミングの一形態であるため、StreamIt における多くの言語機構は Overlay GHC により記述できると考えられる。現時点において、Overlay GHC の設計では、生成されたオーバレイネットワークの性能について言及していないが、Overlay GHC プログラムを StreamIt のコードにコンパイルするような技術は検討に値すると思える。

7　今後の課題

Overlay GHC はいまだ開発途上にあるプログラミング言語である。その仕様を向上・改善していくためには、この言語によるプログラミングの実績を更に積んでいく必要がある。既存のオーバレイネットワークをこの言語で実装する試みの一例として、分散ハッシュテーブルの一種である Kademlia[14] を Overlay GHC で実装する試みが始まっている。その他の分散ハッシュテーブルや様々な分散アルゴリズムがそれに続く予定である。

トレーサブルネットワークへの応用に関しては、実際に Overlay GHC で並列推論エンジンを記述し、ツールキットに組み込むことで、インシデントの発生から推論と調査の反復による問題解決に至る流れにおける有用性を検証したい。同時に、筆者らは仮に OG と呼んでいるスクリプティング言語の開発を計画している。この言語は1 対1 で Overlay GHC に対応する。StreamIt とは別のレベルで、Overlay GHC は非ノイマン型アーキテクチャのための高級アセンブリ言語である（KL1 が並列推論マシンのシステム記述言語だったことを想起して頂きたい）。環境情報学系の学生を主とする少数のプログラマに Overlay GHC を使ってもらうトレーニングの経験を通して、筆者らは、論理/並行プログラミングに慣れていない（あるいはプログラミング自体に慣れていない）プログラマの負荷を軽減するために、より高水準な言語が必要ではないかとの感触を抱いている。

8 むすび

この稿では、並行論理プログラミング言語GHCに基づくオーバレイネットワークプログラミング言語として開発途上であるOverlay GHCを紹介した。また、その応用可能性として、特にトレーサブルネットワークのための並列推論エンジンへの適用を論じた。

P2P、グリッドコンピューティング、そしてマルチコア・プロセッサの普及に見られるように、コンピュータシステムにおける並列実行環境が高度化かつ広範囲で利用可能となってきていることに伴い、ソフトウェアシステムにおいて我々が取り扱うべき並行性の水準が、今、急激に上昇している。そのような状況下で、並行性の表現と並列実行を指向する数々のプログラミング言語が提案・検討されているが、Overlay GHCの研究開発はそうした試みの一つと位置付けられる。その中でも、Overlay GHCはオーバレイネットワークを鳥瞰的な視点から素早くプログラミングし、論理ネットワーク全体を一つのプログラムとして記述できる特徴を持つ。

現在、この言語を用いた実際のオーバレイネットワークプログラミングの例として、Kademliaが実装されつつある。また、その他の分散ハッシュテーブルや様々な分散アルゴリズムをこの言語で実装する試みが進められている。同時に、トレーサブルネットワークに向けたツールキットにおける並列推論エンジンの記述を進める予定である。

参考文献

1 K. Ueda, Guarded Horn Clauses. PhD thesis, The University of Tokyo, 1986.

2 K. Ueda and T. Chikayama, "Design of the kernel language for the parallel inference machine", The Computer Journal, Vol.33, 1990 Dec.

3 S. -O. Nyström, "Guarded Horn Clauses, application and implementation", Tech. Rep. Report 41, Uppsala Universitet, Nov. 1987.

4 K. Saito, "TGHC: Timed guarded horn clauses", in Proceedings of the Ninth TRON Project Symposium (International), pp.122-134, IEEE Computer Society Press, Dec. 1992.

5 R. D. Schlichting and F. B. Schneider, "Fail-stop processors: An approach to designing fault-tolerant computing systems", ACM Transactions on Computer Systems, Vol.1, Aug. 1983.

6 L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem", ACM Transactions on Programming Languages and Systems (TOPLAS), Vol.4, Jul. 1982.

7 渕一博監修，古川康一・溝口文雄共編，並列論理型言語GHCとその応用．共立出版，1987.

8 松村量，高山啓，高木祐介，加藤紀夫，上田和紀，“分散言語処理系DKLICの設計と実装”，日本ソフトウェア科学会第19回大会論文集，Sep. 2002.

9 B. T. Loo, T. Condie, J. M. Hellerstein, P. Maniatis, T. Roscoe, and I. Stoica, "Implementing declarative overlays", in Proceedings of ACM Symposium on Operating System Principles (SOSP), Oct. 2005.

10 J. Armstrong, "The development of Erlang", in SIGPLAN International Conference on Functional Programming, 1997.

11 G. Agha, Actors: a model of concurrent computation in distributed systems. MIT Press, 1986.

12 K. M. Kahn and V. A. Saraswat, "Actors as a special case of concurrent constraint (logic) programming", ACM SIGPLAN Notices, Vol.25, Oct. 1990.

13 W. Thies, M. Karczmarek, and S. Amarasinghe, "StreamIt: A language for streaming applications", in Proceedings of the 2002 International Conference on Compiler Construction, Apr. 2002.

14 P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the XOR metric", in Proceedings of IPTPS02, Mar. 2002.

斉藤賢爾（さいとうけんじ）
情報通信セキュリティ研究センター
レーサブルネットワークグループ専攻
研究員　博士（政策・メディア）
分散システム、実時間システム、コンピュータコミュニケーション