プログラミング－言語・基礎・実践－ 5－5
(1991. 11. 21)

並列リダクションにおける引数評価時の決定戦略

干場美佳子
日本アイ・ビー・エム株式会社 東京基礎研究所

あらまし

関数型プログラミング言語のをコンビネータのグラフリダクションによって並列実行する際の引数の評価時を決定するための戦略を提案する。従来の方法では完全遅延評価を実現するためにstrictness analysisを行なって早い時点で評価を行なっても影響のない引数を見つけ出し、それらすべてを事前評価していた。しかし、strictness analysisは十分に並列性を抽出できない。期待される並列性をconcurrency-levelと呼ばれるパラメータで表し、これを用いてより並列性の高い時点で引数を評価を行なうための戦略を提案する。

Argument Evaluation Strategy for Parallel Graph Reduction

HOSHIBA, Mikako
IBM Research Tokyo Resaerch Laboratory

Abstract

Graph reduction of combinator is one way to execute functional programming languages. For parallel execution we have to decide which arguments should be evaluated and when they should be evaluated. To keep advantages of fully lazy evaluation, we should find arguments that can be evaluated before reduction. In earlier implementation strictness analysis was used to find the arguments, and all strict arguments were evaluated as earlier as they could. But strictness analysis can extract low concurrency. I propose strategy to decide which argument should be evaluated and when they sould be evaluated. In this strategy we can compare possibility of concurrent execution by using new paramater concurrency-level and we can evaluate arguments with higher concurrency.

1 はじめに

関数型プログラミング言語は、副作用がなく部分式が独立している。式の評価の方法には、関数の呼び出し前に引数を評価する方法と、評価をできる限り遅らせ、必要になるまで行なわない完全遅延評価とがある。関数の呼び出し前に引数を評価する方法は、引数評価を同時に行なうことで容易に並列化できる。しかし、すべての引数の値が実際に必要となるわけではないため、単純にすべての引数を並列に評価したのでは計算量の点で無駄が多く、もっとも効果的な並列実行の方法とはいい難い。一方、完全遅延評価は、不必要な式の評価を行なわないため、全体としての計算量はもっとも少ない。しかし、並列実行の可能性は非常に低い。

本稿では、静的な解析によって、完全遅延評価の計算量に影響を及ぼさずに並列性を引き出す方法を提案する。具体的には、関数型プログラミング言語の実行方法としてコンビネータによるグラフリダクションを採用する。並列実行は引数の並列評価によって実現する。strictness analysis を用いて計算量に影響を与えない引数を見つけ、それらの引数のうち、いつ、どの引数を評価すべきかを決定する戦略についての検討を行なう。

関数型言語のコンビネータリダクションによる実行は Turner によって提唱され[3]、実際のリダクションマシンもいくつか提案されている。コンビネータリダクションの逐次実行では、グラフリダクションによって式の共有を行ない、最外最左のコンビネータに関してリダクションを行なっていくことで簡単に完全遅延評価が実現できる。

strictness analysis[1] は式の停止性に関する理論である。これを利用して完全遅延評価の場合にも必ず評価される部分式を見つけ出すことができる。こうした引数の評価はいつ行なっても全体の計算量に影響を与えない。したがって、コンビネータの書き換え規則に strictness analaysis を施し、計算量に影響を与えない複数の引数を見つけ出し、それらを並列に評価することで、完全遅延評価の利点である最小限の計算量で並列実行が可能になる。

strictness の情報は式の値の必要性に関する情報だけで、その式を評価することで得られる並列性の情報を含まない。しかし、これまでの方法では、strictness analaysis で得られた情報によって事前評価可能な引数を見つけ、それらの引数をすべてリダクション前に評価していた。strictness が並列性に関する情報を持たないにもかかわらず、並列実行のための引数の評価時の決定に用いられていた。そのため、効果的に並列実行できない場合がある。strictness の情報に加えて、実行の並列性に関する情報が必要である。

本稿では並列性を高めるために、期待される引数評価の並列性を concurrency-level と呼ばれるパラメータで表し、その値によって引数の評価時を決定する戦略、concurrency-level prediction を提案する。この戦略を用いることで、strictness analysis で見い出した事前評価可能な引数を、より並列性の高い時点で評価することができる。

2章ではコンビネータとリダクションについて説明し、3章では strictness analysis だけを用いた並列実行とその問題点について述べる。concurrency-level prediction について4章で述べ、5章に実験結果を示す。6章では5章の結果をふまえ、今後の課題を検討する。

2 コンビネータとリダクション

2.1 スーパーコンビネータ

本稿では、コンビネータとしてスーパーコンビネータ [2] を用いている。

関数定義は、そこから自由変数を含まない部分式をパラメータとして取り出すことによってコンビネータとその書き換え規則に変換される。この時、自由変数を含まない最大の部分式 (maximal free expression:mfe) を取り出すことによって得られるものがスーパーコンビネータである。

関数 fa をスーパーコンビネータに変換する過程を図1 (a) に示す。fa の定義中から各引数に対して順に mfe を取りだす。まず、zに対する mfe である $(if\ (fb\ x\ y))$,$(+\ y)$ をパラメータ i,p として取り出し、自由変数を含まないラムダ式 $(\lambda ipz.i\ (p\ z)\ z)))$ を得る (1) 。この式を FA2 と名付け、対応する書き換え規則を得る (5) 。FA2 を用いると fの定義は (2)

$f = \lambda xyz.if\ (g\ x\ y)\ (+\ y\ z)\ z$

(a) mfe

$f = \lambda xy.(\lambda ipz.i\ (p\ z)\ z)))\ (if\ (g\ x\ y))\ (+\ y)$ (1)

$= \lambda xy.$ F2 $(if\ (g\ x\ y))\ (+\ y)$ (2)

$= \lambda x.(\lambda gy.$ F2 $(if\ (g\ y))\ (+\ y)$ (3)

$= \lambda x.$ F1 $(g\ x)$ (4)

F2 $ifc\ py\ z \rightarrow ifc\ (py\ z)\ z$ (5)

F1 $gx\ y \rightarrow$ F2 (IF $(gx\ y)$) $(+\ y)$ (6)

F $x \rightarrow$ F1 (G x) (7)

(b) mrfe

$f = \lambda xy.$ F2 $(g\ x\ y)\ y$

$= \lambda x.$ F1 $(g\ x)$

F2 $gxy\ y\ z \rightarrow$ IF $gxy\ (+\ y\ z)\ z$

F1 $gx\ y \rightarrow$ F2 $(gx\ y)\ y$

F $x \rightarrow$ F1 $(g\ x)$

図1:スーパーコンビネータの生成

で表される。yについても同様の操作を行ない(3)、コンビネータF1を得る(6)。FA1を用いたfの定義には、もはや自由変数は存在しない(4)。これが定義した関数faに対応するコンビネータのFAの書き換え規則となる(7)。こうした一連の操作によって、faは3つのコンビネータに変換される。書き換え規則中のFBは関数fbに対応するコンビネータを表す。スーパーコンビネータは、グラフリダクションによって完全遅延評価されるという性質を持つ。この性質は、パラメータとして自由変数を持たない部分式のうち、リダクション可能な最大のもの(maximal reducible expression: mrfe)を取り出しても変化しない。図1 (b)はmrfeをパラメータとした場合のスーパーコンビネータの生成過程である。zに関するmfeは$(if\ (fb\ x\ y)),(+\ y)$であるが、ifは3引数、+は2引数であり、これらの式はリダクションが実行できない。従って取り出すべきmrfeはリダクション可能な部分式$(fb\ x\ y)$,yの2つである。これらのmrfeをmfeの場合と同様にパラメータ化することでコンビネータが得られる。

二つの場合に生成されるコンビネータの書き換え規則を比較すると、mrfeではmfeに比べて引数が書換え後に関数部分に代入されることが少ない。こうした引数は、値が定まるためにはさらに引数を必要とする式(high order function)である。事前評価可能な部分を見つけ出すためstrictness anlaysisを行なう場合、関数(コンビネータ)部分が引数では、そのhigh order functionの引数に対する解析が難しい。strictness analysisの効率の点からは、high order functionが形成されにくいmrfeによってコンビネータを生成することが望ましい。

2.2 グラフリダクション

本稿では、リダクションの方法としてグラフリダクションを用いている。

グラフリダクションは、二進木で表された式をコンビネータの書き換え規則にしたがって順に変形していくことで計算を行う。最外最左のコンビネータの書き換え規則が適用され、引数のうち規則の右側に二ヶ所以上現れるものはコピーされるのではなく、ポインタによって共有される。これによって、一つの部分式は常に一つの二進木としてしか存在せず、評価が一度行なわれれば、それを共有していた他の部分式は、その式を評価することなく評価後の値を共有し、利用することができる。これによって、完全遅延評価が実現される。図2はグラフリダクションによる部分式の共有の例である。

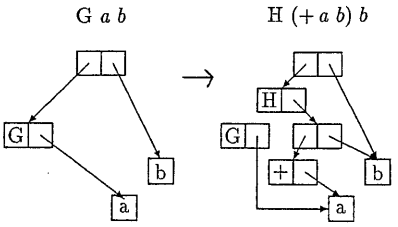


図2：グラフリダクションによる部分式の共有

3 並列リダクション

3.1 完全遅延評価と strictness analaysis

コンビネータの式をグラフリダクションで評価する場合、基本的な評価戦略は完全遅延評価である。式の評価は値が必要となった時に一度だけ行なわれ、それ以後はその値を用いられる。そのため、必要最小限の計算量で式の評価が行なわれるという利点を持つ。

しかし、並列処理を考えた場合には完全遅延評価は有効な方法ではない。式全体の評価は最外最左のコンビネータのリダクションによって逐次的に進み、それ以外の部分式、つまりコンビネータの引数の評価は値が必要となった時に初めて行なわれる。したがって、リダクションによる書き換え後に最外最左にコンビネータの現れる可能性のないコンビネータが、式の評価のために値を評価する必要がある引数を複数持つ場合にだけ並列実行が可能である。つまり、2引数の数値演算子、論理演算子といった組み込みの演算子の引数について並列性だけしか得られない。完全遅延評価での並列実行は組み込み演算子の引数に関してのみ行なわれ、評価する式に潜在している並列性を十分に引き出すことはできない。完全遅延評価は、引数の並列評価による並列化には不適当である。

計算量を最小にしたままで、引数の並列評価によってコンビネータリダクションを並列化するために strictness analysis が用いられる。strictness analaysis は計算の停止性に関する理論であるが、これを利用して後の計算の過程で必ず評価される部分式を見つけ出すことができる。式全体の停止性を支配する部分式は必ず評価される部分式であり、いつ評価を行なっても、式全体を評価するための計算量に影響を与えない。strictness analysis によって式の評価のために必ず値が必要になる引数を見つけ出し、それらをリダクションの実行前に並列に評価することができる。strictness analaysis を用いることで、完全遅延評価では得られなかった定義された関数、コンビネータの引数の並列性を引き出すことができる。

図1 (b) のコンビネータの書き換え規則を見ると、FA2 では IF が第1引数の値を必要とし、FA2 の第1引数 $fbxy$ は strict である。また IF が第2、第3引数、のいずれかの式を評価することから、両方で strict な引数 z が strict である。条件が真なら $(+\ y\ z)$ が評価され、+は2つの引数の値を必要とし、偽なら z が評価される。これによって FA2 の2つの引数 $fbxy$,z が事前評価可能となり並列に評価される。このように、関数定義から部分式を取り出してスーパーコンビネータに変換し、strictness analysis を行なうことで、fa に内在していた $(fb\ x\ y)$ と z の並列性を取り出すことができる。

3.2 strictness analysis の問題点

strictness analaysis を行うことで、定義された関数、コンビネータの引数を事前評価し、完全遅延評価では得られなかった並列性を引き出すことができる。しかし、単に strict な引数をすべて事前評価しただけでは式に内在する並列性を十分に引き出すことはできない。

多引数関数からコンビネータを生成する過程では、各引数に対して順に mrfe を取り出し、その引数が関与しない部分式の事前評価の可能性を引き出す。しかし、一つの関数が複数のコンビネータに分解されるため、各コンビネータが high order function を形成する。それらのコンビネータに対して順に関数の引数が与えられるために、関数の引数の評価がリダクション実行間に分散することがある。こうした評価の分散は、常に並列性を高めるわけではない。

たとえば、図3 (a) の関数 f は2つの式 $(g\ x)$,$(h\ y\ z)$ の和で表され、+の引数の並列評価が期待される。f は3つのコンビネータに変換

される（図3 (b)）。この書き換え規則にstrictness analysisを行なう。ただし、Gの引数は事前評価可能でないとする。F2の規則では、書換え後+の2つの引数、gx,$(hy\ z)$が必ず評価される。したがって、F2の引数ではgx,hyの2つがstrictであり、事前評価が行なわれる。同様に、F1ではgxが事前評価される。fに引数としてa,b,cを与え、この規則にしたがって評価を行なった場合の、式の変形の過程を図3 (c-A)に示す。+の引数の評価時に注目すると、第一引数はF1の実行前に(1)、第二引数はF2の実行前後(2)(3)にそれぞれ評価され、並列評価は起こらない。これは、第一引数となる部分式$(g\ a)$がF1の実行前に作られるのに対して、第二引数は部分式$(\mathrm{H}\ b)$がF2の実行前に作られ、F2実行後に第二引数$(h\ b\ c)$の値を得るために必要な引数が揃うために起こる。

strictness analysisによって得られた事前評価可能な引数は、評価可能なもっとも早い時点で評価される。そのため、strictな部分式の評価はその式の引数が揃った時点で行なわれ、並列評価可能な式が共通の引数を含まないとき、評価はコンビネータのリダクションの前後に分けられ、並列性を得ることはできない。strictness analysisだけでは共通の引数を持たない部分式の並列性を引き出すことができない。

4 Concurrency-Level Prediction

関数定義中の共有引数を持たない部分式の評価が並列に実行できないという問題は、strictness analysisが単に式の値の必要性の情報に過ぎないにも関わらず、この情報によって事前評価可能な引数を決定すると同時に、その引数をいつ評価するのかを決定していることによって引き起こされる。strictnessの情報はその引数が必ず評価されることを示しているだけで、そこに存在する並列実行の可能性は明らかではない。strictな引数を可能なかぎり早い時点で評価するのではなく、並列性を考慮して引数の評価時を決定すべきである。

式の持つ並列性

(a) 関数定義

$f\ =\ \lambda xyz.\ +\ (g\ x)\ (h\ y\ z)$

(b) コンビネータ

$\mathrm{F}\ \ x \rightarrow \mathrm{F1}\ (\mathrm{G}\ x)$

$\mathrm{F1}\ \ gx\ y \rightarrow \mathrm{F2}\ gx\ (h\ y)$

$\mathrm{F2}\ \ gx\ hy\ z \rightarrow +gx\ (hy\ z)$

(c) リダクションによる変形過程

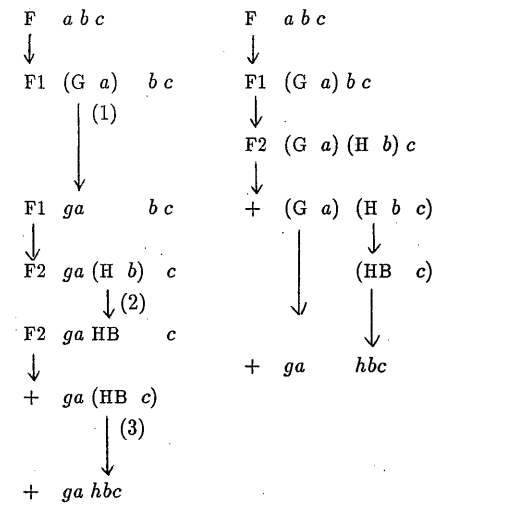


図3：strictness analysisの問題

を表すパラメータとして concurrency-level を提案する。concurrency-level は書き換え規則の両側について、事前評価可能な引数の数にもとづいて計算される値である。もっとも簡単には並列評価可能な引数、つまり、事前評価可能かつ評価済みでない引数の数である。したがって、concurrency-level の計算には strictness analysis の結果とコンビネータ間での引数の引渡しに関する情報が必要である。

引数の引渡しの情報は引数がコンビネータに渡される時の状態の情報である。定義された関数に対応するコンビネータの引数としては任意の式を考えなければならない。しかし、書き換え規則の適用によってのみ現れるコンビネータの引数の状態は、それ以前のコンビネータの引数の事前評価の状況によって異なり、すでに評価済みの場合がある。事前評価された引数が次のコンビネータにそのままの形で渡される場合、その引数は並列実行に貢献しない。また、書き換え後の引数部分に同一の形の式が複数できる場合には、それらは共有されるので、並列実行される引数としては1つのものとして扱う。

concurrency-level の定義と事前評価を行なう引数を決定する方法は以下の通りである。

(1)**Concurrency-level**

書き換え規則の

右側　strict かつ未評価の引数の数

左側　異なる形の strict かつ未評価の引数の数

(2)**Concurrency-level の計算と評価時の決定方法**

1. strictness analysis を行ない、事前評価可能な引数を見つける。
2. 関数定義に対応するコンビネータから順に、
   書き換え規則の左右の concurrency-level を計算する
   if concurrency-level: 左 > 右 then
   strict な引数は事前評価のされる。
   他のコンビネータの引数のうち、事前評価される引数がそのまま渡されるものに評価済みの印をつける。
   endif
   この操作をコンビネータが無くなるまで繰り返す

次の関数 ga について GB の引数がすべて strict である場合の concurrency-level を計算する。

$$ga = \lambda wxyz.\, gb\ (+\,x\ y)\ (+\,x\ y)\ (+\,w\ z)$$

$$\begin{aligned} &\text{GA}\quad w\ x\ y &&\rightarrow\quad \text{GA1}\ w\ (+\,x\ y)\\ &\text{GA1}\ w\ pxy\ z &&\rightarrow\quad \text{GB}\ pxy\ pxy\ (+\,w\ z)\end{aligned}$$

まず、strictness analysis を行なう。GA では GB の3つの引数が strict であり、GA1 の3つの引数も strict である。GA では右側で GA1 の2引数が strict であり、GA の2つの引数も strict である。GA の左右の concurrency-level は左:3、右:2 であり、リダクション前の方が並列実行の可能性が高い。したがって、GA の引数は事前評価される。GA による書換え後、wは評価済みのため GA1 の第1引数に評価済みの印をつける。次に GA1 の concurrency-level を見ると、左は wが評価済みのため2、右では GB の3つの引数が事前評価可能であるが、第1、第2引数が共有された1つの式のため concurrency-leval は2である。

strictness analaysis だけでは並列性を引き出せなかった図3 (b) に concurrency-level prediction を行なう。H,G について情報がない場合、strict な引数は、F2 の規則の右側では+の2引数、したがって左では gx, hyである。同様にして各コンビネータの strict な

引数が見つけられる。下の表は各コンビネータの両側のstrictな引数(args)とconcurrency-level(level)を示したものである。

	左		右	
	args	level	args	level
F	x	1	(G x)	1
F1	gx	1	gx,(H y)	2
F2	gx,hy	2	gx,(hy z)	2

concurrency-level は F:0→1,F1:1→2,F2:2→2 であり、どのコンビネータでも減少しない。したがって、事前評価はまったく行なわれない。この規則したがったリダクションを実行すると（図3 (c-B)）、+の2引数の評価の分散せずに並列に評価される。

5 実験結果

並列リダクションマシン・シミュレータ上で実験を行い、strictness analysis と concurrency-level prediction の実行を比較した。

シミュレータはKCLで実現されている。シミュレータは、与えられた関数定義をコンビネータとその書き換え規則に変換する。また、与えられた式をグラフリダクションによって評価する。各命令は、それぞれ固有のstep数をを持ち、シミュレータはグラフリダクションの実行時に実行step数を数える。実行step数は実行時間に対応する値である。CPUは無限個を仮定する。

分割征服法によってyからxまでの数の和を求める関数$sum0$を用いて実験を行なった。$sum0$の定義、それを変換して得られるコンビネータの書き換え規則を図4 (a)に示す。$sum0$はyからxまでの数を2つのグループに分割し、それぞれの和を計算し、それらの値を足し合わせることで全体の和を求める。したがって、部分和の計算の並列実行が期待される。

図4 (b)は、一階のstrictness analysis（high order functionの引数の解析を行なわない）を行い$sum0$を実行した結果を示すグラフである A1 ではstrictness analysisのみ行いstrictな引数をすべて事前評価した。B1 ではconcurreny-level predictionを行なって引数の評価時を決定した。y軸は実行step数である。x軸の$x-y$は引数の差であり、$sum0$が再帰的に呼び出される回数を支配する。$x-y>0$の時$sum0$は$x-y+1$回再帰的に呼び出される。

strictness analysisだけを用いた場合には実行step数が関数の呼びだし回数に比例して増え、並列実行が行なわれていないことがわかる。それに対して、concurrency-level predictionを行なった場合にはstep数は階段状になり、関数の呼びだし回数が2^nになると増加する。+の2引数の並列評価が行なわれて$sum0$の呼び出しの二進木ができ、この木が一段増えるたびに実行step数が増えていることがわかる。

このように、concurrency-level prediction は、strictness analysisだけを持ちいた場合には引き出せなかった+の2引数の並列性を引き出している。

図4 (c)は$x-y=7$についてのデータである。同時に評価された部分式の最大数を見ても、concurrency-level predictionが$sum0$の呼び出しの二進木を作って+の引数の並列評価を行なっているのに対し、strictness analysisがこれを行なっていないことがわかる。

ほとんどの項目でconcurrency-level predictionがstrictness analysisだけの場合よりも良い結果を示しているが、リダクションの回数だけは逆の結果を示している。この差は、引数の共有によって起きたもので、増加分はすべて書き換え規則I$x \rightarrow x$を持つIコンビネータのリダクションである。

計算量を最小化するためは、共有される部分式の評価は一度だけ行ない、その後は結果を参照しなければならない。グラフリダクションではルートセルをリダクションの結果での書換えることで実現される。しかし、部分式のリダクションの結果が数などの場合には結果が木にならず、ルートセルの書き換えが行なえない。そのため、ルートセルをIコンビネータと結果の値で書き換える方法をとっている。これによって、共有部分式のリダクションは最悪でもIコンビネータのリダクション1回で済む。

strictな共有引数は、A1で事前評価されるが、B1では事前評価されない場合があり、これがリダクションの回数の差として現れている。

strictness analysis を high order function の引数まで行なった場合との比較を図4 (d)に示

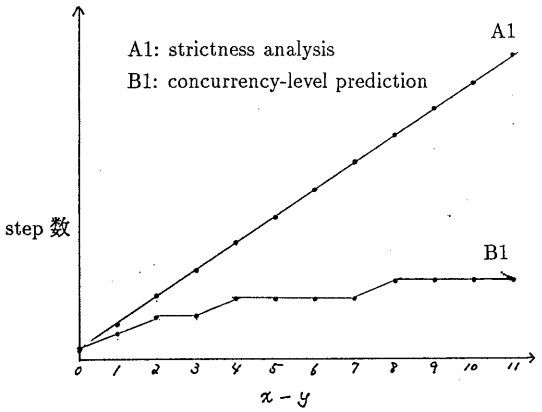
(a) 関数とコンビネータ

$sum0 = \lambda xy.\, if\ (=\ x\ y)\ x$
$\quad (sum2\ x\ (floor\ (+\ x\ y)\ 2)\ y)$
$sum2 = \lambda xmy.\, +\ (sum0\ x\ (1+\ m))\ (sum0\ m\ y)$

SUM0 x → SUM1 x (SUM2 x)
SUM1 $x\ sx\ y$
→ IF $(=\ x\ y)\ x\ (sx\ (floor\ (+\ x\ y)\ 2)\ y)$
SUM2 x → SUM3 (SUM0 x)
SUM3 $sx\ m$ → SUM4 $(sx\ m)$ (SUM0 m)
SUM4 $sxm\ sm\ y$ → $+\ smx\ (sm\ y)$

(c)$x - y = 7$

	A1	B1
step 数	49352	15266
評価した引数の数	47	27
同時に評価された部分式の最大数	3	9
リダクションの回数	109	123
使用したセル	231	231
read	3115	2948
write	1746	1651



(b)$sum0$ の実行 step 数

(d)strictness analysis のレベルによる step 数の比較

strategy		A	B
strictness	1	49352	15721
analysis	2	50042	48785

$x - y = 7$
strictness analysis 1:一階 2:high order function

図4 ：concurrency-level prediction の効果

す。high order function まで解析を行なった場合、concurrency-level prediction でも並列実行が起こらず、実行 step 数は strictness analysis のみの場合とほぼ同じである。SUM4 の書き換え規則中で+の第2引数の high order function 、sm が (SUM0 m) であるために、yも strict になり SUM4 の concurrency-level が $3 \rightarrow 2$ となり、+の2引数の並列実行が起こらないためである。

high order function の引数として strict な引数が加わることで、リダクション前の concurrency-level が増え、引数評価の分散が起こっている。

6 今後の課題

concurrency-level prediction を用いることで、strictness analaysis で得られた事前評価可能な引数を、並列実行の可能性を考慮して評価することが可能となった。concurrency-level として、単純に並列評価可能な引数の数を用いても改善が見られたが、concurrency-level prediction の目的は、式に内在する並列性を可能な限り引き出すことである。そのためには、concurrency-level の計算に際して次のような要素を考慮していく必要がある。

6.1 引数の共有

コンビネータの引数がリダクション実行後に共有される場合、その後の実行中に複数の式が共有された値を必要とし、評価が競合する場合がある。一つの式の評価は一度以上行なわれることはない。そのため、後から評価しようとした方は実行を中断し、評価終了を待って結果の値を得る。競合による中断再開のコストが大きい場合には、共有される引数の評価が事前に行ない、評価の競合を避けることが望ましい。concurrency-level を計算する際に共有引数の重さを非共有引数に比べて大きくするか、あるいは、書換え規則の右側の共通の引数を持つ引数の重みを軽くすることで共有引数が事前評価されやすくすることができる。

6.2 High order function

書き換え規則の右側の式中で、同時評価可能な引数を形成する部分式が関数部分にコンビネータの引数を持つとき、その引数はリダクション前の評価によって high order function になる引数であり、リダクション実行後に、さらにいくつかの引数が与えられる。定義された関数自体が high order function を返す場合を除いて、こうした high order function は一つの多引数関数から生成されたのコンビネータ群の1つであり、次のコンビネータへ渡される引数となる式を作る役目を果たす。そのためコンビネータの引数のとしての大きな計算量を持たない場合が多い。従って、high order function となる引数については、concurrency-level の計算の重みを軽くするべきである。

6.3 引数評価の分散化

リダクションの前後で2つ以上の引数が並列評価可能な場合、前後それぞれに複数の引数を評価できる場合がある。次の関数 f について並列実行の可能性を考えてみる。

$$f = \lambda xy.\, g\ (h\ y)\ x\ (h\ x)$$

$$\begin{array}{lll} \mathrm{F}\ \ x & \rightarrow & \mathrm{F1}\ x\ (\mathrm{H}\ x) \\ \mathrm{F1}\ \ x\ hx\ y & \rightarrow & \mathrm{G}\ (\mathrm{H}\ y)\ x\ hx \end{array}$$

G,H の引数がすべて strict な時、F,F1 の引数もすべて strict である。concurrency-level prediction を行なうと、F:1 → 2 ,F1:3 → 3 となり事前評価は行なわれない。しかし、リダクションの流れは

$$\mathrm{F}\ ab \rightarrow \mathrm{F1}\ a\ (\mathrm{H}\ a)\ b \rightarrow \mathrm{G}\ (\mathrm{H}\ b)\ a\ (\mathrm{H}\ a)$$

である。F1 のリダクションを考えた場合、引数の評価を前後どちらか一方で行なうよりも、F1 の実行前に a, b を評価し、リダクション後に (H a),(H b) を評価する方が効果的である。

このように、すべての事前評価可能な引数を事前評価する／しない、と決めてしまうのではなく、事

前評価する引数と事前評価しない引数に分け、評価を分散させることで並列性を高めることができる。

こうした評価の分散のためには、リダクションによる式の変形によって起こる部分式の共有の情報が不可欠である。複数の関数にまたがった場合は引数の形態の特定が難しいが、1つの関数を分解して得られたコンビネータ群の中では、共有される引数と引数間の依存関係によって部分式の共有情報を得ることができる。

concurrency-level を書き換え規則の前後ではなく、リダクションの間の値として、依存関係のある式の評価／未評価によって重みを変えて計算を行ない、もっとも並列性の高い引数評価の組合せを見つけ出すことが重要である。

評価を分散させる方針は、共有部分を持つ部分式の評価が同時に起こらないようにすることである。そのためには、事前評価可能な引数のうち、共有される引数、事前評価可能な引数の部分式となる引数の評価を優先させるべきである。

7 おわりに

コンビネータリダクションで事前評価を行なう引数を決定する方法として、concurrency-level prediction を提案した。実験を行い、従来の strictness analysis のみを用いた場合には得られなかった並列性を引き出せることを示した。

式に内在する並列性を引きだし、さらに大きな並列性を得るためには、先に述べた点からの改良を行なっていく必要がある。

参考文献

[1] Clack,C., Peyton Jones,S.L *Strict analysis - a practical approach, Functional Programming and Computer Architecture, LNCS201, Springer-Verlag, 1985*

[2] Peyton Jones,S.L. *An introduction to fully-lazy super combinators, Combinator and Functional Programming Language, LNCS242 Springer-Verlag, 1985*

[3] Turner,D.A. *New Implementation Technique for Applicative Languages, Software Practice and Experience vol.9, 1979*

[4] 干場美佳子 並列リダクションにおける引数評価時の決定法, 第41回情報処理学会 全国大会