数, queen1 の第 2 引数, および : の第 2 引数である. queen のいずれの経路の第 1 引数も

$g_4 = (0 \quad 7 \quad 7 \quad 7 \quad 7 \quad 7 \quad 7 \quad 7 \quad 7 \quad 7 \quad 7 \quad 11)$

$g_5 = (0 \quad 3 \quad 3 \quad 3 \quad 3 \quad 3 \quad 3 \quad 3 \quad 3 \quad 3 \quad 3 \quad 11)$

の g_4, g_5 というモードが入っていることがわかっている. また, queen1 の第 2 引数も同様である. 7(nil) で呼ばれる場合は, : は 11(誤り) を出すので, 注目している : は第 2 引数の conc に 3 を要求する. よって conc の中の : に 3 が要求され, conc の右辺の : は常に第 2 引数に 3($\omega : \perp \in$) を要求することがわかる.

4 関連研究

Finne [4]らは [3]で行われている手法に沿って遅延評価を行なうか必要かあるか否かを静的に解析した. さらに Spineless G-Machine 命令を遅延評価用と複数の先行評価用に生成した. そして, どの版の命令を選択するかを, 翻訳時に静的に決定することと実行時に動的に決定することの効率の比較の実験を逐次処理系の上で行なった. 結果は多くの場合静的に決定することを選択する方がメモリ参照が少ないことがわかった.

本研究はこの結果を考慮し, 翻訳時に静的に使用する命令を決定する.

5 評　価

現在, 第 3 節のアルゴリズムを処理系への組み込み中なので, ここで図 2のプログラムを手で解析し, それに基づいて, コード生成して, AP1000 の 64 台構成の上で実行した. その結果を表 1に示す. 実行時間は 5 回実行させてその平均を取った.

解析結果を組み込んだコード生成では 2% ほど速度が向上していることがわかる. これは, 外部クロージャの参照の回数がほぼ 4 分の 1 に減っているためだと考えられる. このため, 通信回数が減るだけでなくプロセスがサスペンドする回数も減っていて, これが速度の向上につながっている.

なお, 同じプログラムを 1 台で動かした実行時間は, 7.75 秒なので, 64 台用いたにもかかわらずスピードアップは 16 程度にとどまっている.

6 おわりに

遅延評価型関数型言語において静的解析の解析結果を並列実行の効率化にどう反映させてコード生成を行なうかに関して実験した.

表 1　評価 (AP1000 64PE)

	解析なし	解析あり
実行時間 (秒)	0.451	0.442
ゴール生成数	676	676
外部クロージャの参照	1008	368
サスペンド回数	1526	845

今後は, 第 3 節のアルゴリズムを処理系に組み込んで, 解析力が十分かどうか, 実用的な時間で解析が終了するかするかなどを調べると共に, もっと大きな問題に関しても実験を行なう予定である.

参考文献

[1] Bird, R. and Wadler, P.(武市正人 訳): 関数プログラミング, 近代科学社, 1991.

[2] Brus, T., Eekelen, M.C.J.D. van, Leer, M. van, Plasmeijer, M.J.: Clean - A Language for Functinal Graph Rewriting, FPCA'87, LNCS 274, pp.364-384, 1987.

[3] Geoffrey L. Burn: *Lazy Functional Languages: Abstract Interpretation and Compilation*, Pitman in association with MIT Press, 1991, Research Monographs in Parallel and Distributed Computing.

[4] S. O. Finne and G. L. Burn: Assessing the Evaluation Transformer Model of Reduction on the Spineless G-machine, *Proc. of the Conf. on Functional Programming and Computer Architecture*, pp. 331–340, ACM, 9–11 June, 1993, Copenhagen.

[5] Jones, M. P.: An Introduction to Gofer, University of Oxford, 1991.

[6] Jones, SL, P.: Implementing lazy functional languages on stock hardware: the Spineless Tagless G-machine, *Journal of Functional Programming*, 1992.

[7] 木村康則, 近山隆: 並列論理型言語 KL1 の多重参照管理によるガベージコレクション, 情報処理学会論文誌 Vol. 31, No. 2, pp. 316-327, 1990.

[8] 小野諭: 関数型プログラムのストリクト性関連解析と最適化技術, 情報処理学会論文誌, Vol. 29, (1988), No. 8, pp. 862–871.

[9] 田中哲朗: 疎結合並列計算機用の関数型言語処理系, 日本ソフトウェア科学会第 10 回大会論文集, pp. 325-328, 1993.

[10] 田中哲朗, 山本貝英, 武市正人: 疎結合並列計算機上の遅延評価型関数型言語処理系の性能評価, 情報処理学会研究会資料 94-PRG-18, pp. 41-48, 1994.

[11] Philip Wadler: Strictness analysis on non-flat domains (by abstract interpretation over finite domains), *Abstract Interpretation of Declarative Languages*, pp. 266–275, Ed. S. Abramsky and C. Hankin, Ellis Horwood, 1987.

ンタを輸出する．輸入した側でそのクロージャが必要になったら，その輸出テーブルに対する read_var メッセージを送る．そのメッセージを受けとったノードは元のクロージャを評価するためのタスクを生成し，クロージャの評価結果が WHNF になった際に，要求したノードの輸入テーブルにクロージャのコピーを送る．

2.3 単一参照クロージャに対するコード生成

STG の逐次実行の際，クロージャの単一参照性が保証されると，そのクロージャは update frame を積まない形にコンパイルできる．また，配列の要素を変更した新しい配列を作る際にコピーせずに破壊的代入で実現できる．

単一参照性の解析は並列実行の際の通信量の削減にも役に立つと考えられる．単一参照のクロージャを輸出する際には輸出輸入テーブルを介さずに WHNF と同様にコピーを送ることができる．こうすると，後でこのクロージャを必要にした際の read_var, write_import の 2 回のメッセージを減らすことができる．ただし，WHNF の輸出と同様に値が必要でなかった場合のクロージャの輸出のコストが増大することもある．

単一参照のクロージャは図 2ではイタリック体で表した部分である．

2.4 必須クロージャに対するコード生成

単一参照でないクロージャであっても，必須なクロージャは先行評価して弱頭部正規形に直してしまえば，そのままの形でコピーできる．図 2中のボールド体で表したものが，単一参照ではない必須なクロージャである．

spec の第 2 引数として現れるクロージャは必須であるが，これを先行評価すると並列性が得られないので，解析の際には考慮する．

必須であることがわかった式はクロージャを作らず先行評価する形にコンパイルする．元々の STG では，ベクターリターンを用いて，caller 側が update function を知る必要があったため先行評価を実現するのが難しかったが，我々の処理系では callee 側にまかせたので効率を起こさずに実現できた [10].

3 抽象実行による静的解析

必須性解析の為に図 3の抽象領域を使用する．

リストの頭部は $\{\omega \leq \top\}$ で表され，それぞれ具象領域の値の「情報がない」，「情報が定まる」と対応する．リストの尾部に Wadler の 4 点のリスト抽象領域 [11]の要素を置く．Wadler の 4 点のリスト抽象領域は $\{\bot \leq \infty \leq \bot\in \leq \top\in\}$ という構造をしており，それぞれ具象領域のリストの，「情報が無い」，「コンスがあるがリストの長さは定まらない」，「リストの長さは定まるが，その要素に不定のものを含む」，「リストの長さ，その要素両方とも定まる」と対応している．

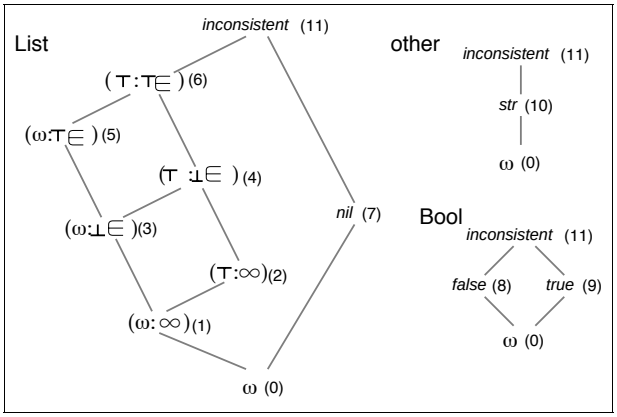

図 3 必須性解析用抽象領域

抽象解釈の枠組として計算経路解析 [8]を使用する．計算経路解析の特徴はモードという抽象領域上の 1 階の 1 引数関数を使用して，ある関数が抽象関数値のある値を要求された場合の引数への要求の伝播を解析していることである．モードは

$$\begin{array}{rcccccccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ g_1 = (0 & 0 & 10 & 0 & 10 & 0 & 10 & 11 & 11 & 11 & 11 & 11) \\ g_2 = (0 & 0 & 0 & 3 & 3 & 6 & 6 & 11 & 11 & 11 & 11 & 11) \\ g_3 = (0 & 0 & 0 & 7 & 7 & 7 & 7 & 11 & 11 & 11 & 11 & 11) \end{array}$$

のようなもので，本稿では g_i で示す．例えば g_2 は 0,1,2 を要求されたら 0 を返す関数である．関数 $f(v_1, v_2, \ldots, v_n)$ はモードを使って $\langle$ v_1 のモード，v_2 のモード，…，v_n のモード $\rangle$ というタプルの集合として表される．例えば

$$\mathcal{A}[\![: v_1\ v_2]\!] = \{\langle g_1, g_2\rangle, \langle g_1, g_3\rangle\}$$

と : を表す．：に 4($\top$: $\bot$ $\in$) が要求されると，$\{\langle g_1 4, g_2 4\rangle, \langle g_1 4, g_3 4\rangle\} = \{\langle 10, 3\rangle, \langle 10, 7\rangle\}$ となりその第 2 引数は 3($\omega : \bot\in$) もしくは 7(nil) が要求されることがわかる．

たとえば，n-queens の各関数が計算経路解析で抽象領域関数に抽象化されているときに，conc の定義の右辺の : の第 2 引数の conc の特化したモードを求めるには以下のように解析する．

conc が呼び出されているのは queen の第 1 引

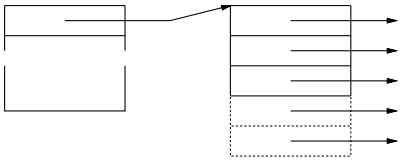

図 1　クロージャの構造

図 2は n-queens のプログラムであるが, 下線を引いた部分の実行でクロージャを作る必要がある.

クロージャは複数回参照される場合があるが, 同じ式を何度も評価しないために評価した後は上書き (update) される. これは, クロージャの Info Table の Standard エントリで update frame と呼ばれる 4 ワードの領域をコントロールスタックに書き込んでおくことで実現される.

2.2　並列性の導入

遅延評価型関数型言語の枠組の中でも, 複数の引数を必須とする組み込み関数に起因する並列性は導入できるが, それ以外の並列性や投機的な並列実行のために組み込み関数 spec を用意している.

spec は, spec f x = f x となる高階関数で f x の評価とx の評価を別のプロセッサで行なう. x の評価は弱頭部正規形 (Weak Head Normal Form 以下 WHNF) になるまでしか行なわない. 同じプロセッサ内で並列に実行するための specLocal も用意している.

spec は以下のように実現されている.

1. メッセージのヘッダ (export_goal) を準備してから別プロセッサで実行したいクロージャの Info Table の Export エントリ引数の level を 0 にして呼び出す.
2. クロージャの実体をメッセージバッファにコピーし, クロージャ自身は輸入テーブルへの間接ノードとする.
3. メッセージを他のノードに送る
4. 受けとったノードではクロージャの実体をメッセージバッファからヒープに移し, タスクキューにつなげる
5. クロージャを評価して WHNF になったら, 元のノードの輸入テーブルへ WHNF になったクロージャをコピーして送る (write_import)

```
main :: Dialogue
main = appendChan stdout (show (go 10))
                  abort done
go n = count (queen1 0 (gen n) [] [] []) 0
gen 0 = []
gen (n+1) = (n+1) : gen n
count [] n = n
count (h:t) n = count t (n+1)
queen1 n [] [] l o = specLocal (l:) o
queen1 n [] (h:t) l o = o
queen1 n (p:u) c l i =
  spec (check1 n l p 1 u c l)
       (queen1 n u (p:c) l i)
check1 n [] t d u c b o =
  if (2<=n) then
    queen (conc u c) [] (t:b) o
  else
    queen1 (n+1) (conc u c) [] (t:b) o
check1 n (p:r) t d u c b o =
 if ((t==(p+d)) || (t==(p-d)))
 then o
 else check1 n r t (d+1) u c b o
queen [] [] l o = specLocal (l:) o
queen [] (h:t) l o = o
queen (p:u) c l i =
  check l p 1 u c l (queen u (p:c) l i)
check [] t d u c b o =
  queen (conc u c) [] (t:b) o
check (p:r) t d u c b o =
 if ((t==(p+d)) || (t==(p-d)))
 then o
 else check r t (d+1) u c b o
conc [] y = y
conc (w:x) y = w : conc x y
```

図 2　n-queens のプログラム

5 の前に元のノードでそのクロージャが必要となったら, そのタスクは輸入テーブルのサスペンドリストに加わる. その輸入リストへ write_import メッセージが届いたらそこでアクティベイトされる.

spec で直接外部に出されるクロージャではなく, そこから間接的に参照されるクロージャの場合は実体は移さず, 輸出テーブルを確保しそれを指すポイ

疎結合並列計算機上の関数型言語実行の効率化

Improving efficiency of parallel functional languages on loosely coupled multiprocessor systems

田中 哲朗† 下國 治† 武市 正人†
Tetsurou TANAKA Osamu SHIMOKUNI Masato TAKEICHI

†東京大学工学部
Faculty of Engineering, University of Tokyo

概 要

遅延評価型の関数型言語において, 大域的な必須性解析や単一参照解析はメモリの参照回数が少ない効率的なコードを生成するのに有効であることが知られている. 疎結合並列計算機上で並列実行させる場合は, それに加えて通信量やサスペンド回数を減らすことができ, 効率的な実行が可能になると考えられる. 本研究では AP1000 上の並列 Gofer 処理系上で実験的に必須性解析, 単一参照解析の結果に基づくコードを生成をして, その評価を行なった.

1 はじめに

遅延評価型の関数型言語は, 高階関数, 遅延評価の性質を使ってアルゴリズムを簡潔に記述できる点, λ 算法に基づき数学的に定義されていて副作用を持たないためプログラム変換の対象として扱い易い点などから近年注目を集めている. しかし, 遅延評価はヒープのアクセス回数が多く, 先行評価と比較して速度やメモリ消費量の点で劣るという問題点がある. そのため, 遅延評価型の関数型言語でも必須性解析 (Strictness Analysis) を行ない先行評価が可能な式はあらかじめ評価する方式が研究されている [2].

関数型言語の特徴の 1 つとして, 並列性を内在するため並列実行に適している点がある. 我々は, 関数型言語 Gofer [5]を並列化し, 疎結合並列計算機 AP1000 の上で並列に実行する処理系を実現した [9]. 処理系を最適化する過程で並列実行の際に必須性解析, 単一参照解析を生かしたコードの生成法に関するアイデアが得られた.

本稿では, そのコード生成法と解析アルゴリズムを提案し, それに基づいた解析とコード生成を手で行ない評価した結果を示す, この解析に基づくコード生成が実際に疎結合並列計算機上で有効であることが確められた.

2 遅延評価型関数型言語の並列実行

2.1 Spineless Tagless G-machine

遅延評価型関数型言語の逐次実行モデルとしてグラフ簡約に基づくモデルがいくつか提案されている. 中でも Spineless Tagless G-machine(STG) [6]は Stock hardware 上で効率的に実行されるように工夫されたモデルである. 我々の並列 Gofer 処理系も, 元の Gofer 処理系で使われた G-machine ではなく STG を拡張したコードを生成し, 元の処理系よりも高速な逐次実行速度を実現した [9] [10].

遅延評価の際, 関数の引数に現れる関数適用 (例えば `f (g x)` の `g x`) のような式は, 必要になるまで評価されない. このような式を STG ではクロージャと呼ばれるヒープ上の構造で実現している. 図 1は我々の処理系で用いるクロージャの構造である.

クロージャを評価する際に用いる標準エントリだけでなく, GC の際に用いる Evacuate, Scavenge エントリ, 並列実行用に加えられた Export, Import エントリなど, クロージャに対する操作を細かく設定することが可能になっている.